USENIX

conference

proceedings

2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet

San Jose, CA, USA
July 7, 2006

Sponsored by
The USENIX Association

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Proceedings of the 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet

San Jose, CA, July 7, 2006

For additional copies of these proceedings contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Phone: 510 528 8649
FAX: 510 548 5738
Email: office@usenix.org
URL: http://www.usenix.org

The price is $15 for members and $20 for nonmembers.
Outside the U.S.A. and Canada, please add
$6 per copy for postage (via air printed matter).

Thanks to Our Sponsors

at&t

Microsoft® Research

ISBN 1-931971-46-3

USENIX Association

Proceedings of the 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI '06)

July 7, 2006
San Jose, CA, USA

Conference Organizers

Program Chair

Steven M. Bellovin, *Columbia University*

Program Committee

Harald Alvestrand, *Cisco*
Dan Boneh, *Stanford University*
Jon Crowcroft, *Cambridge University*
Anja Feldmann, *Technische Universität München*
John Ioannidis, *Columbia University*
Balachander Krishnamurthy, *AT&T Labs—Research*
Chris Morrow, *UUnet*
Vern Paxson, *ICIR/ICSI*
Niels Provos, *Google*
Eric Rescorla, *Network Resonance*
Tara Whalen, *Dalhousie University*

Steering Committee

Clem Cole, *Intel, USENIX liaison*
Dina Katabi, *Massachusetts Institute of Technology*
Balachander Krishnamurthy, *AT&T Labs—Research*
Ellie Young, *USENIX*

Publicity Chair

Lakshminarayanan Subramanian, *Intel Research, Berkeley/New York University*

The USENIX Association Staff

External Reviewers

Jia Wang
Robin Sommer

2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet

July 7, 2006
San Jose, CA, USA

Friday, July 7, 2006

Message from the Program Chair

Anyone who lives on the Internet deals with unwanted traffic. It may be the obvious—who among us doesn't get spam? It may be crippling, as the victims of distributed denial of service attacks have learned. Or it may be hidden—it is a truism among those who monitor live networks that there is an amazing amount of inexplicable traffic present, packets that couldn't possibly exist in that location but somehow do. SRUTI addresses all of these concerns, and more. Only 9 of 20 papers were accepted. The workshop is purposely kept small, to encourage interaction and conversation. These papers are not a complete answer, of course, but there's a quote from the Talmud that applies: "It is not your part to finish the task, yet you are not free to desist from it."

Steven M. Bellovin, Columbia University
Program Chair

The Rising Tide: DDoS from Defective Designs and Defaults

Richard Clayton

University of Cambridge, Computer Laboratory, William Gates Building,
15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom
richard.clayton@cl.cam.ac.uk

Abstract

We consider the phenomenon of distributed denial of service attacks that occur through design defects (and poorly chosen defaults) in legitimately operated, entirely secure systems. Particular reference is made to a recently discovered "attack" on stratum 1 Network Time Protocol servers by routers manufactured by D-Link for the consumer market, the latest example of incidents that stretch back for decades. Consideration is given to how these attacks might have been avoided, and why such failures continue to occur.

1 Introduction

Currently, when considering distributed denial-of-service (DDoS) attacks, the image is of a wicked individual commanding an army of "zombies" to emit specially crafted Internet Protocol (IP) packets so that an innocent victim is overwhelmed by the incoming traffic. The effect (a system that is unable to communicate effectively) can be just the same when there is a sudden rise in legitimate traffic, a "flash crowd" or "the Slashdot effect". Between these two extremes lies a further type of denial-of-service, where design defects in hardware or software create a substantial amount of traffic that is not malicious *per se*, but not necessarily legitimate either.

This paper is about these design error "attacks". There will typically be a slow increase in intensity; starting with small amounts of traffic from a handful of sources, but growing into a significant nuisance when the traffic is arriving from thousands or millions of endpoints. One might think of zombie-based DDoS attacks as directing a firehose onto the victim; likening slashdotting to being drowned in a flash flood; and the design errors creating an inexorably rising tide. In all three cases the result can be drowning.

In Section 2 we present a number of historical examples of DDoS attacks caused by design errors, then in Section 3 we describe a brand-new example in some detail. In Section 4 we divide the general problem of design errors into categories and discuss the type of mitigation and/or avoidance that could be appropriate. Finally in Section 5 we summarise and draw some fairly gloomy conclusions.

2 Historical Examples

2.1 HOSTS.TXT

The original ARPANET mapped hostnames to network addresses by means of a flat-file database (`HOSTS.TXT`) which was centrally updated and then distributed to every participating machine. The system was replaced (by about 1987) with the Domain Name System (DNS) that we still use today. A commonly cited justification for the need for the new system is the ability to delegate the responsibility for changes – it could take several days for administrative staff to add a new host to the central file.

More significantly, in the current context, these changes were batched together and an announcement made when a new file version was available. There would then be a flash-crowd as every ARPANET host attempted to fetch the new copy [15]. This effect was getting ever worse ($O(n^2)$) as the number of downloaders increased and the size of what they were downloading increased in proportion as well.

2.2 Root Nameserver Traffic

In January 2001 Brownlee *et al* studied [2] the traffic at the 'F' root nameserver, one of the machines that handles queries for data at the top-most level of the DNS. They found one host sent the same query 2 112 962 times in an hour (587 queries per second) and that 12%–18% of queries for A records were for IP addresses (viz: for values that were already resolved). Their overall conclusion was that a significant part of the system load could be directly attributed to traffic that did not follow the DNS protocol.

Although their analysis identified flaws in many different types of software (including some email viruses that made DNS queries) they drew attention to three specific issues from Microsoft – which their market share makes especially relevant.

Firstly, names from Microsoft's local networking protocols, such as WORKGROUP, were turning up in root nameserver queries as `.workgroup`. Secondly, Windows 2000 systems defaulted to generating "dynamic update requests" (reflecting reallocation of IP addresses by DHCP on local networks) and some of these updates were – entirely pointlessly – being sent to the root nameservers (in one case, a single machine sent 15 000 requests to 'F' in a day). Thirdly, when a DDoS attack on a router made Microsoft's nameservers unavailable (in defiance of Best Practice, they were both on the same subnet) requests for the top-level records for Microsoft domains reached 25% of the overall query load.

2.3 Netscape's Parallel Downloading

Until Netscape was released in October 1994, the standard way of fetching a web page was to download the file, parse it to locate further items – such as embedded images – and download those one at a time. Netscape ran faster by fetching the images in parallel, using a user-settable number of connections, defaulting to 4. Initially, this was extremely controversial, because it caused many more processes to be running on web servers, and this was considered to be tantamount to a DDoS attack (see, for example, [6] and related articles in the `comp.infosystems.www.misc` newsgroup in January 1995). In time, it became apparent that servers can be configured to cope, and this has become the standard way of fetching web pages (although note that in HTTP/1.1 multiple items can be fetched over a single connection).

Similarly, `qmail`, Dan Bernstein's Mail Transfer Agent (MTA) has attracted criticism (and robust defence) for its parallel delivery of email messages (which defaults to 20 parallel streams, but is often set higher) [4]. The community has dealt with this by ensuring that other MTAs have (configurable) limits on the number of incoming connections per host that they are prepared to accept.

2.4 Mojo Nation

Mojo Nation was an "emergent file store" that ran from July 2000 until February 2002 [17]. It was a distributed data haven that used digital cash ("mojo") to pay for storage services. Over 100 000 users downloaded the software, although there were only ever 100–600 persistently connected nodes and a maximum system size of about 10 000 nodes. The designers were surprised to find that sites were connected for short periods (less than an hour) and many sites never returned after connecting to the network once.

Unfortunately, the system had a single "original introduction" server which provided a list of existing nodes to new nodes that wished to join in. When Mojo Nation was mentioned on Slashdot [12] as an improvement on Napster and Gnutella, the server was overwhelmed by the large number of new users and failed to return responses to users for several days.

2.5 Network Time Protocol Servers

In 2000 the University of Delaware changed the name of their NTP server from `time.ultimeth.net` but three years later they found that DNS requests for it were increasing over time. A Windows program called 'NetTime' had hard-coded their system identity, along with a number of other well-known servers. In September 2003 they removed the domain from the DNS, but traffic was still arriving several weeks later, probably because of flawed DNS caching [8].

In October 2002 a web server at Trinity College Dublin was overwhelmed by HTTP traffic generated by 'Tardis', a shareware program for setting the time on Windows machines. A web-based time access system had been added for users whose firewalls blocked NTP traffic, and users were requesting synchronisation as often as once per minute. The resulting load, when it eventually became too much for the server to handle, was about 420 requests per second and rising. The software was not patched for some time, but reconfiguring the server to provide a bogus time led to a substantial fall in requests [13].

In May 2003 the University of Wisconsin – Madison found itself at the receiving end of continuous large-scale flood of inbound Simple Network Time Protocol (SNTP) packets [16]. The rate exceeded 280 000 packets per second. The source was determined to be consumer equipment (routers) manufactured by Netgear, of which there were about 700 000 in the field. These devices sent SNTP packets requesting the time and, if there was no answer within one second (!), asked again. Having received a satisfactory answer no further request would be made for one minute (or longer periods up to 24 hours, depending upon the product). Clearly, with this algorithm, once the connection to the SNTP server became congested, any systems that managed to receive answers would be contributing to the traffic again before the remaining systems had been lucky enough to get an answer.

In July 2003, two stratum 1 NTP servers run by

CSIRO in Australia were moved to "secret" IP addresses to avoid being accessed at a rate of 6 000 packets per second by approximately 85 000 routers manufactured by SMC Networks [18, 9]. The devices only accessed the clock a few times a day, but when the server operators decided this traffic was excessive and blocked it, the device firmware increased the request frequency to twice a minute. A contributing factor was that many of the routers were behind firewalls that discarded the NTP response, so these were already making requests at the faster rate.

2.6 Dynamic DNS

Many ISPs provide their customers with "dynamic IP addresses" that may be changed to a new value on the next connection, or at a later time. However, customers may wish to run servers, such as a web hosts, at invariant hostnames. Hence DNS entries must be updated to track the changing IP address. A number of companies provide a service whereby an HTTP transaction can trigger a DNS update, and it has become commonplace for cable modems and ADSL routers to be configurable to automatically perform an update whenever their IP address changes.

In October 2005 Dynamic Network Services Inc. (who operate dynamic DNS domains such as `dyndns.org`) announced that they would be failing to action HTTP requests containing the string `client/1.0`, which they believed to be sent by some models of D-Link device [7]. They had 1.4 million customers, but were receiving 21 million invalid updates per day, and estimated that approximately 10 000 devices were generating 25% of their traffic load.

3 D-Link & Stratum 1 NTP

In the late summer of 2005 Poul-Henning Kamp discovered that a very high proportion of the traffic to `gps.dix.dk` (the stratum 1 NTP server he operated in Denmark) consisted of obsolete NTP version 1 requests. All of the machines who should have been "chiming" against his system were using the current, NTP version 4, request packets, so the specious traffic was easy to quantify. He concluded that he was suffering a DDoS attack from zombie-infested computers and asked for help in tracking down the "botnet" he thought was responsible.

He collected the traffic in `tcpdump` format over a number of days and made it available for analysis to the present author. On a typical day, 1 Nov 2005, 3.19 million NTPv1 packets arrived (37 per second) from 276 256 unique IP addresses. Collating these IP addresses by AS (ISP) made it clear that if the source of the traffic, which uses UDP for transport, was spoofed, then the perpetrator had made an excellent job of mimicking the actual usage of IP addresses. It was far more likely that the source addresses were valid. After identifying a source IP address in the UK and contacting the user, it was possible – having eliminated a number of other possibilities – to identify that the traffic had been sent by a "DI-624" wireless router manufactured by D-Link.

An identical DI-624 was purchased, which arrived with v2.42 firmware dated 31 Mar 2004. In its default state (with no NTP server specified by the user), the device used a preset list of 63 NTP servers (including `gps.dix.dk`). Every 2.2 seconds the device made a DNS request to resolve a name selected from this list (in an unpredictable order, often resolving the same host several times in a row), and every 30 seconds it issued an NTPv1 request packet. Although initially the NTP requests were to the most recently resolved server, later requests could immediately precede the re-resolving of the relevant hostname. Hence the device was issuing many pointless DNS requests, ignoring the DNS conventions on the validity of cached results, and – by sending out two NTP requests a minute – ignoring the NTP conventions as well.

It will be noted that there is some discrepancy between the average arrival rate of NTPv1 packets at `gps.dix.dk` (averaging one per 2 hours per source) and the rate at which they are being sent (about once every 30 minutes to any particular NTP server). This can be partly explained by the use of dynamic IP addresses, but may well be because other D-Link products (several models were found to be using stratum 1 NTP servers) retry slightly less often.

Of the 63 NTP servers in the v2.42 firmware list, 52 remain (April 2006) in the `isc.org` canonical list of stratum 1 servers [10]. Of these, 24 are "Open Access" and do not require users to notify the owners of usage; 6 are "Open Access" but require notification; 5 are "Restricted Access" but don't require notification; 15 are "Restricted Access" and require notification; and the last 2 are "Closed Access". Although "Open Access" servers "may be used without restrictions by any client in any location", they are also documented to have particular "Service Areas", showing which particular countries or networks they are "intended to serve". More importantly, the standard "Rules of Engagement" (`http://ntp.isc.org/bin/view/Servers/RulesOfEngagement`) require clients that access stratum 1 servers to be in synchronisation subnets of two or more systems and to themselves be providing service to more than 100 other clients. Quite clearly, the D-Link DI-624 falls well outside all of these criteria.

The latest version of the firmware available for UK devices (in April 2006) is v2.59b3 (dated 30 Nov 2005). The server list is markedly changed from v2.42 (but identical to the list in v2.53 – 20 Apr 2005), with 31 entries removed (including `gps.dix.dk`) and 36 added to give a total of 68, all bar one of which is on the current canonical list. However, although 30 are now classed as "Open Access" the other systems still have restrictions and the two "Closed Access" systems remain. Also, although the server list has changed, the dynamic behaviour – synchronising the time every 30 seconds – is unaltered.

4 Addressing DDoS by Designs

We can identify several types of activity that give rise to the DDoS examples cited above:

Service Discovery is obtaining the identity of systems that can provide a service. For example, the ARPANET `HOSTS.TXT` file provided the identity of every machine on the network, and the Mojo Nation "original introducer" provided a list of active nodes.

Service Access is the use of a service, and the DDoS occurs from there being just too many users; specifically, in the various NTP examples, because access is being made by inappropriate systems.

Broken Systems are generating traffic which is just plain wrong and this causes DDoS effects, either first-hand, as in the systems identified by Brownlee *et al* sending hundreds of DNS queries per hour (or per second), or second-hand where the simultaneous failure of the `microsoft.com` DNS servers significantly increased the load on the root servers.

In addition, in a handful of the examples, such as the parallel usage of HTTP and SMTP streams, the "DDoS" has turned out, in the long term – with appropriate default behaviour – to be liveable-with.

4.1 Mitigation

Mitigation of DDoS caused by design issues is usually achieved through the replication and distribution of servers. For example, clients access DNS servers that cache results so as to avoid repetitive fetching from authoritative sources. This is combined (or should be) with the replication of those authoritative sources and the "root servers". In a like manner, Mojo Nation addressed its problems by replicating its servers.

For web content, companies like Akamai (`www.akamai.com`) now provide "content distribution" networks, specialising in serving popular content from servers that are hosted at ISPs all over the world. As servers fail or become overloaded they redirect requestors to other locations where the same content will be available – though they do this at the expense of reducing the timeout on DNS data to a few minutes or seconds, thereby adding load to that system.

The time servers also operate a distributed system but, crucially, it is based on convention rather than any on-the-wire protocol. The "stratum 1" servers are connected to atomic clocks or receive timing signals from GPS satellites. They are accessed by "stratum 2" servers that fetch the time from several stratum 1 machines, analyse the timestamps and round-trip duration, and thereby estimate the correct time. Similarly "stratum 3" servers access stratum 2 servers to get an only slightly less accurate time value. End-user machines, at the bottom of the pyramid, should be accessing stratum 3 servers.

The end-user devices have several ways of discovering local (stratum 2 or 3) NTP servers. They could use DHCP to obtain the IP addresses (the option code is 42 and was described in RFC2132 [1]); they could use the `pool.ntp.org` project – which maintains a list of almost 600 time servers and provides appropriately chosen random selections in response to DNS queries; or the user could be asked to manually configure a server name.

However, as was clear from Section 2 above, a number of manufacturers have chosen to obtain a list of stratum 1 time servers (these lists are the easiest to locate), have failed to read up on the usage restrictions, have placed these hostnames into their firmware, and have then shipped tens of thousands of units. They have also, on occasion, compounded their error by generating requests at unreasonable rates and failed to "back off" after a connection failure.

This has left the NTP servers with a complex mitigation problem. The requests consume processing power and bandwidth – whether answered or not – and this impacts the timing accuracy for legitimate users. Filtering the traffic at a network ingress point may be impractical (in the D-Link case, the undesirable traffic was easy to recognise since it was NTPv1, but other examples were more complex). The latest version of the time protocols [14] does contain a "kiss-of-death" response, used to tell clients to stop their requests, but implementations based on older documentation will naturally take no notice.

If the servers change their name then this affects their legitimate users, who cannot now locate the system. In principle, the name changing could be done by providing different DNS results to different requestors. But the distributed nature of the DNS could mean that a remote stratum 2 server (which should be given service) resolves the stratum 1 server name using a recursive DNS server that is also employed by hundreds of thousands of consumers.

A different solution is the use of an out-of-band authorisation system. The clients present credentials (or a compelling reason) to the authorisation system and it then provides a single-use server name which can be resolved to give the server address. There are obvious trade-offs between agility (moving the server around to prevent unauthorised access) and inconvenience to legitimate clients – that have to revisit the authorisation system. This system, albeit manually operated and with the IP address provided directly, is what is currently operated by the Australian National Measurement Institute to control access to the NTP servers formerly hosted by the CSIRO [18].

4.2 Education

It is reasonable to ask why there appears to be so much repetition in the examples cited above. The NTP servers seem be encountering problems on a regular basis, and the DNS traffic problems described by Brownlee *et al* are similar to those studied by Danzig *et al* almost a decade earlier [5].

The NTP incidents indicate that the high-level, conceptual, design of the system is poorly understood by the calibre of people who are currently implementing clients. This is worrying, because many aspects of the suite of Internet Protocols depend upon showing restraint, and the need for that restraint is not always immediately apparent. This may help to explain why, when flaws are suspected – as in the Netscape and `qmail` examples – the reaction from the developer "establishment" can be quite heated.

However, community-level design review remains relatively uncommon, and so the lack of comprehension that problems are possible, and the increasing ease of systems development by "just anybody", could well mean that we see ever more DDoS from design flaws. Many of the highest users of bandwidth (such as video and peer-to-peer file-sharing) avoid TCP and roll their own UDP-based schemes, paying limited attention to congestion issues. This is a key driver for the provision of protocols such as DCCP [11], which aim to provide an easier-to-use alternative to UDP that will – by design – prevent the clueless from DDoS-ing the Internet.

4.3 Economics

It used to be that by the time one had built a TCP/IP stack the ethos of restraint had been absorbed ('security through inculcation'), but if the next generation of stack builders try to obtain more performance for themselves and do not bother to back off in the face of congestion then we could see cascading network collapse. This, along with several of the historical examples, is essentially the "tragedy of the commons" and so it is attractive to consider an economic approach to fixing these problems. For example, after the SNTP problem, Netgear made a donation of $375,000 to the University of Wisconsin – Madison [3]. Clearly, this was not widely enough reported (or a sufficiently large sum) to register with the other manufacturers whose designs have caused similar DDoS problems.

Voluntary donations are never likely to be a long-term solution, and a financial penalty approach would need underpinning by the courts. Unfortunately, the legal system is not currently well suited to this type of problem. It is not especially easy to issue proceedings in other countries; individual damage may be small and so "class actions" would be required – posing difficulties in some jurisdictions; and it remains unlikely that local regulators (or consumer protection officials) will be interested in deeply technical matters, where apparently minor errors cause significant impact to other sites elsewhere in the world.

Another economic approach is to consider where the incentives are. When a design causes a DDoS, why should any of the millions of "attackers" fix their part of it? End-users can be frightened into removing zombies from their machines by suggesting that the security of their own data (and the likelihood of keeping their ISP account) is at risk. But no ISP is, realistically, going to disconnect customers for sending 120 NTPv1 packets an hour; and whether or not a wireless router knows the correct time is pretty much irrelevant to most people.[1] The experience of Trinity College, Dublin was that returning the wrong time can reduce traffic (though users may well have discarded the software rather than reconfiguring it), but it also showed that software, once shipped, can be just as hard to get updated as firmware.

5 Conclusions

We have shown that DDoS is not restricted to activity channelled through insecure zombies, but can also occur through requests that would be a tolerable nuisance in small numbers, but in the mass can have devastating results. These requests come from systems whose design just doesn't scale, or which contain bugs, or that encapsulate misapprehensions about what is acceptable behaviour. These problems have, to large extent, been addressed by replicating resources, but many still continue to pose significant challenges for the "attacked" machines.

[1] Which leads one to wonder why manufacturers bothered with NTP in the first place!

This type of DDoS is unlikely to be fixed by users (or ISPs on their behalf) "taking responsibility" for the traffic, since from their point of view they have done nothing terribly wrong. The "attacked" machines will get little mileage out of creating lists of "attackers" and demanding their disconnection.

What is most disturbing is the sense of déjà vu one gets from examining these events. The same types of error seem to be recurring, suggesting a failure to properly educate system developers to avoid designs that can DDoS. Looking at the problem from a security economics point of view is equally gloomy. The incentives are not suitably aligned to address the problem, and where manufacturers have paid for their mistakes this not yet been effective at preventing others from falling into the same trap – and there seems little short-term likelihood of a legal approach being effective at the network-scale necessary.

Finally, it is noteworthy that the systems that are being "attacked" are seldom managing to detect the undesirable traffic at an initial low volume (or even at an intermediate stage when its nature is bound to be apparent), but are only becoming aware of an issue when the problem has become acute. In 1992, Danzig *et al* recommended regular "policing" of server logs to detect anomalies and contact defective systems. This is still not common practice – but surely if one lives in a tidal area, keeping an eye on the water level is just commonsense.

Acknowledgments

The identification of the D-Link traffic was made possible with the assistance of Nick Webb and his family. Richard Clayton is currently working on the spamHINTS project, funded by Intel Research.

References

[1] S. Alexander, R. Droms: DHCP Options and BOOTP Vendor Extensions. RFC2132, IETF, 1997.

[2] N. Brownlee, kc claffy, E. Nemeth: DNS Measurements at a Root Server. IEEE Global Telecommunications Conference, 2001, (GLOBECOM '01), 25–29 Nov 2001, vol 3, pp. 1672–1676.

[3] Division of Information Technology, University of Wisconsin – Madison: Serendipity and hard work pay off. 21 Apr 2004. `http://www.doit.wisc.edu/news/story.asp?filename=322`

[4] Common Vulnerabilities and Exposure (CVE) List: Denial of service in Qmail by specifying a large number of recipients with the RCPT command. Candidate (under review) CVE-1999-0144. 7 Jun 1999.

[5] P. B. Danzig, K. Obrackza, A. Kumar: An Analysis of Wide-Area Name Server Traffic: A study of the Internet Domain Name System. ACM SIGCOMM Computer Communication Review, 22(4), Oct 1992, pp. 281–292.

[6] S. Desai: Re: FAQ: Netscape Communications and our products. comp.infosystems.www.misc, 14 Jan 1995. `http://groups.google.com/group/comp.infosystems.www.misc/msg/e194375a44e90ea8`

[7] Dynamic Network Services Inc.: User-Agent Block: client/1.0. 31 Oct 2005. `http://www.dyndns.com/about/company/notify/archives/useragent_block_client10.html`

[8] D. Gibson: Why we have abandoned the UltiMeth domain. Sep 2003. `http://www.ultimeth.com/Abandon.html`

[9] HPCwire: Network Devices Almost Take Down Atomic Clock. 11 Jul 2003. `http://www.taborcommunications.com/hpcwire/hpcwireWWW/03/0711/105537.html`

[10] Internet Systems Consortium Inc.: Stratum One Time Servers. `http://ntp.isc.org/bin/view/Servers/StratumOneTimeServers`

[11] E. Kohler, M. Handley, S. Floyd: Problem Statement for the Datagram Congestion Control Protocol (DCCP). RFC4336, IETF, Mar 2006.

[12] R. Malda: Forget Napster & Gnutella: Enter Mojo Nation. 9 Oct 2000. `http://slashdot.org/article.pl?sid=00/10/09/1826243`

[13] D. Malone: Unwanted HTTP: who has the time. Usenix ;login 31(2), Apr 2006.

[14] D. Mills: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI. RFC4330, IETF, Jan 2006.

[15] National Research Council Committee on Internet Navigation and the Domain Name System: Signposts in Cyberspace. National Academies Press, Sep 2005, ISBN: 0-309-09640-5.

[16] D. Plonka: Flawed Routers Flood University of Wisconsin Internet Time Server. 21 Aug 2003. `http://www.cs.wisc.edu/~plonka/netgear-sntp/`

[17] B. Wilcox-O'Hearn: Experiences Deploying a Large-Scale Emergent Network. in: P. Druschel, F. Kaashoek, A. Rowstron (Eds.): Peer-to-peer Systems, First International Workshop, IPTPS 2002, Cambridge MA, USA, Mar 2002, pp. 104–110.

[18] M. Wouters: Re: Improved monlist scheme – a bit off topic. ntp:hackers mailing list, 15 Sep 2003. `http://lists.ntp.isc.org/pipermail/hackers/2003-September/000326.html`

Efficient and Secure Source Authentication with Packet Passports

Xin Liu and Xiaowei Yang
Department of Computer Science
University of California, Irvine
{*xinl,xwy*}*@ics.uci.edu*

David Wetherall and Thomas Anderson
Department of Computer Science & Engineering
University of Washington
{*djw,tom*}*@cs.washington.edu*

Abstract

A key challenge in combating Denial of Service (DoS) attacks is to reliably identify attack sources from packet contents. If a source can be reliably identified, routers can stop an attack by filtering packets from the attack sources without causing collateral damage to legitimate traffic. This task is difficult because attackers may spoof arbitrary packet contents to hide their identities.

This paper proposes a packet passport system to address this challenge. A packet passport efficiently and securely authenticates the source of a packet. A packet with a valid passport must have originated from the claimed source. The packet passport system can be incrementally deployed without introducing extra control messages. It also provides incentives for early adoption: a domain that deploys packet passport system can prevent other domains from spoofing its source identifiers. Our preliminary analysis suggests that the packet passport system can be implemented at high-speed routers with today's technologies.

1 Introduction

DoS flooding attacks have posed an increasing threat to the reliability of the Internet. An early study showed that DoS attacks occurred at a rate of nearly 4000 attacks per week [23]. Servers, ranging from critical infrastructure such as the root DNS servers [5], to commercial web portals such as yahoo [20], to personal web sites [14], may all fall victim to those attacks. More recently, DoS attacks have been used for online extortion [8]. If this trend continues, it will significantly undermine the potential of the Internet to support mission critical applications, online commerce, and real-time communications.

A key challenge in combating DoS attacks is to reliably identify attack sources from packet contents. This ability can enable a number of DoS defense and prevention mechanisms. Routers can stop an ongoing attack by filtering packets from the attack sources without causing collateral damage to legitimate traffic. They can enforce source-based resource allocation mechanisms such as weighted fair queueing to prevent attack sources from overwhelming legitimate sources. It's also possible to limit network-layer reflector attacks if attack sources that spoof their addresses are reliably identified. More importantly, the ability to identify the sources of an attack alone may deter future DoS attacks.[1]

It is difficult to reliably identify the source of a packet because attackers may spoof arbitrary packet contents to hide their identities. A number of proposals, such as ingress and egress filtering [13], path-based source address validation [21, 22, 24], path marking schemes [4, 30], and SPM [3] all intend to address this challenge. These proposals can effectively limit source address or path spoofing if they are deployed globally and the routing system is trustworthy. However, a single weak link, e.g., a subverted router, or an undeployed region, allows address or path spoofing.

The goal of our work is to provide an efficient and secure mechanism to authenticate the source of a packet. By efficient we mean that routers can authenticate the source of a packet at line speed; by secure we mean that the identify of a source cannot be forged even if part of the routing system is compromised or has not deployed the authentication mechanism.

Our solution is the packet passport system. A packet passport authenticates the origin of a packet as the packet travels from the source to the destination. Routers at passport checking points can independently validate a passport without trusting the rest of the Internet. A packet passport is cryptographically unforgeable, and can be used by a destination as the proof-of-victim to accuse an attack source, and by routers as a reliable source identifier to enforce policies like precisely filtering packets from an attack source.

Our design only requires light-weight Message Authentication Code (MAC) computation at packet forwarding time and is suitable for high-speed routers. It introduces no extra control messages, and provides incentives for early adoption. ISPs that deploy the pass-

[1] According to [15], today's attacks rarely use spoofed addresses because they are already effective without spoofing. However, this situation may change in the future if source-based automated filtering mechanisms are deployed.

port system can immediately prevent other domains from spoofing the passports of their own hosts. It is in contrast to ingress filtering, with which early adopters cannot prevent other domains from spoofing their addresses.

The rest of this paper is organized as follows. We state our design assumptions and threat model in Section 2. Section 3 describes the design of the passport system. We present a preliminary feasibility evaluation in Section 4. Section 5 compares our work with related work. We conclude in Section 6.

2 Threat Model and Assumptions

We assume that both hosts and routers can be compromised. When hosts or routers are compromised, they may modify, duplicate, or attempt to forge the passports of other sources. The goal of our design is to defend against such attacks. Compromised routers may drop packets or modify packet contents, but we do not address this type of attack.

In this paper, we present an architectural design and assume we can change both routers and hosts software. We use the terms *domain* and *AS* (Autonomous System) interchangeably. For clarity, we describe the key aspects of the design at the domain-level, and omit the detailed operations at the router-level. We also assume that a path-vector routing protocol such as BGP is used for inter-domain routing and each border router has the AS path information for each reachable address prefix.

3 Design

The purpose of a passport is to provide an authentic source identifier that can be used by the network to enforce policies such as filtering or rate limiting, and by destinations as proof-of-victim. An ideal packet passport should satisfy the following requirements:

1. A packet passport is unspoofable. Only the real source can produce passports that claim the source as the origin.
2. Domains should be able to verify packet passports without trusting other domains.
3. A packet passport can be efficiently generated and verified at packet forwarding time. Routers should be able to discard packets with invalid passports as close to the actual source as possible.
4. A valid passport cannot be replayed to send multiple packets. Otherwise, an attacker that intercepts a packet with a valid passport may replay the packet to flood a destination, causing the destination to blame an innocent source for the damage.
5. The passport system can be bootstrapped without much manual configuration or out-of-band communication.

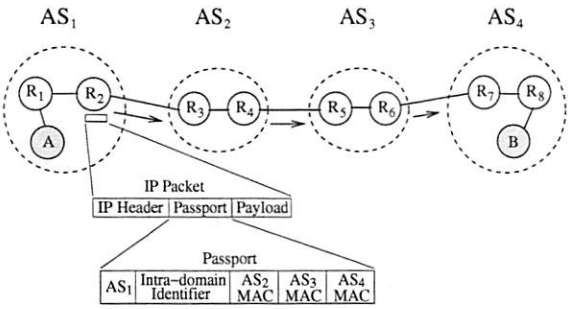

Figure 1: **An example of a packet passport.**

6. The passport system itself is robust against DoS attacks.

We describe how we gear our design to meet these requirements.

3.1 Unforgeable and Efficient Passports

Conceptually, a packet passport can be implemented via digital signatures. A source signs its packets, and routers validate the digital signatures with the source's public key. We discard this approach as digital signatures are computationally expensive to generate and verify. Instead, our design uses a light-weight MAC (such as UMAC [19][2] or HMAC [18]) to implement a packet passport. A packet passport consists of a sequence of AS numbers and corresponding MACs, each computed using a secret key known only by the source domain and a passport-checking domain en route to the destination. These secret keys are distributed as described in Section 3.2. A host sends a packet without a passport. When the border router of the source domain receives the packet, it verifies that the packet is originated from a host within the domain and stamps a passport. When a transit domain receives the packet, it validates the corresponding MAC using the secret key it shares with the source domain. As the key is only known by the source domain and the transit domain, if the MAC matches, it suffices to show that the packet is originated from the source domain. With this design, even if the border routers of a domain are compromised, they cannot forge the passports of other domains.

A packet passport may be implemented as an IP option or as a shim layer. Domains that request passport checking examine the option header or the shim layer, and other domains ignore the passports. Figure 1 shows an example, in which we assume all domains have deployed packet passport system and request passport checking. Let $K(AS_i, AS_j)$ denote the secret key shared between AS_i and AS_j. Suppose the host A in AS_1 sends a packet

[2]The packet ID field in the passport, as shown in Figure 2, can serve as the nonce required by UMAC.

to the host B in AS_4. The border router R_2 stamps a passport that has three MACs. Each MAC is computed using a key $K(AS_1, AS_j)$, $j = 2, 3, 4$, and covers part of the packet as described in Section 3.4. When the packet arrives at AS_2, the border router R_3 uses the secret key $K(AS_1, AS_2)$ to verify that the packet comes from AS_1. Similarly, R_5 and R_7 each uses the corresponding key shared with AS_1 to verify the origin of the packet.

Our design uses a two-level hierarchy for host identification. A source identifier (carried in a passport) consists of a domain identifier (e.g., AS number) and an intra-domain host identifier. The passport issued by a border router only allows a remote domain to verify the origin domain of a packet, not the source host of a packet. We assume that the origin domain uses a separate intra-domain identification mechanism to verify that the host identifier is not spoofed. Our design does not restrict how a domain implements intra-domain host identification. A domain may use any general mechanism to implement the function. It is outside the scope of this paper to discuss these mechanisms, but we sketch a few possibilities.

In one extreme, if a domain can completely prevent source address spoofing inside itself with mechanisms such as fine-grained ingress filtering [13] or SAVE [21], it can use the source address as the intra-domain host identifier. Universal prevention of source address spoofing inside a domain is achievable as a domain is under a single administration. It's also possible that a domain uses the hardware address of a network interface card to identify a host, as commonly used by wireless access points for access control. In another extreme, a domain may use a cryptography-based approach similar to the inter-domain packet passport to identify a host. When a host is connected to the network, it is configured with a secret key shared with the routers of its domain. A host uses this key to authenticate itself inside its domain.

The hierarchical structure of our design improves scalability, but sacrifices security to some extent. A rogue or compromised domain may fake host identifiers in its passports. However, we do not think this is a problem in practice. Routers may allocate resources on a per-domain basis, such as using per-domain weighted fair queueing to allocate its bandwidth. This will discourage a domain from faking host identifiers to obtain more bandwidth share. Similarly, routers on the path may filter or rate-limit all packets from a source domain if the number of hosts that send malicious traffic from the domain exceeds a threshold. This will prevent a domain from launching attacks using spoofed host identifiers.

In our design, each transit domain can independently verify packet passports. This allows routers to drop packets with invalid passports as early as possible and prevents these packets from congesting downstream links. However, it has the side effect of binding a passport to an AS path. In the process of routing convergence, the actual AS path a packet traverses may differ from the one in its passport that is obtained from the source domain's BGP table, and then the packet may be dropped by a transit domain. However, this limitation may not be a serious problem in practice, because even without the passport system there is no guarantee that a packet can reach its destination in case of routing inconsistency. Alternatively, routers may demote packets with invalid passports as low priority packets. When there is no congestion, such packets can still arrive at their destinations.

3.2 Automated Key Distribution

A domain needs to share a secret key with a source domain in order to validate the packet passports from the source domain. The key distribution process must be automated in order to be feasible. In our design, keys are distributed within the inter-domain routing system using a Diffie-Hellman key exchange protocol [9]. We assume each domain has a public/private key pair, and all domains agree on two system-wide Diffie-Hellman parameters p and g.

The key distribution process works as follows. A domain AS_i announces its public key PK_{AS_i} and a public value $d_{AS_i} = g^{r_{AS_i}} \ mod \ p$ to all other domains. r_{AS_i} is a private value chosen by AS_i. The public key and the public value can be piggybacked in a domain's address prefix announcements. With BGP, both PK_{AS_i} and d_{AS_i} can be embedded into one or more prefixes announced by AS_i as optional and transitive path attributes. This design makes the passport system incrementally deployable. To prevent an attacker from modifying the public value d_{AS_i}, AS_i signs d_{AS_i} with its private key.

An attacker may attempt to falsify the public key PK_{AS_i} attached in prefix announcements, i.e. replace it with a different key and use the corresponding private key to sign a fake d_{AS_i} of its choice. This will compromise the Diffie-Hellman key exchange between AS_i and other domains. To prevent this type of attack, our design uses the same mechanism that the routing system uses to prevent the falsification of prefix announcements. For instance, PK_{AS_i} can be certified by a trusted certificate authority (CA) such as ICANN or regional Internet registries as in sBGP [17]. Other approaches such as web-of-trust [29, 33] can also be used to bind PK_{AS_i} to its origin AS AS_i.

The distribution of public keys may be eliminated with Identity Based Encryption infrastructure [6] (IBE). However, we discard the IBE approach because of its rigid key revocation mechanism and its requirement for a single public key generator.

When a domain AS_i receives a prefix announcement from a domain AS_j, it receives d_{AS_j} and vice versa. The two domains then share a Diffie-Hellman secret key:

$K(AS_i, AS_j) = d_{AS_i}^{r_{AS_j}} \ mod \ p = d_{AS_j}^{r_{AS_i}} \ mod \ p$. It is computationally infeasible for an attacker to obtain $K(AS_i, AS_j)$ even if he can eavesdrop the public values d_{AS_i} and d_{AS_j}.

A domain can use a dedicated server such as a route reflector or RCP server [7] to generate r_{AS_i} and d_{AS_i} and to synchronize all the keys on its border routers. If a router reboots and loses its keys, it may obtain them from this dedicated server or other routers in the same domain. It's possible that an entire domain AS_i loses all keys (other than its public/private key pair) due to failures. In this case, after it is back on line, it may acquire the public keys and the public values of other domains from its neighbors, similar to a BGP table transfer after a router reboots. At the same time, it starts announcing prefixes with the latest d_{AS_i} so that the secret key $K(AS_i, AS_j)$ stored in AS_j can be updated.

Distributing keys within the routing system has two primary advantages. First, it allows the passport system to bootstrap. Before keys are properly distributed, a domain cannot send packets with valid passports to cross domains. Distributing keys within the routing system solves this problem. As routing packets are exchanged between adjacent domains, adjacent routers can be configured to accept each other's routing packets without requiring a passport. Second, distributing keys within the routing system allows us to protect the key distribution channel from DoS attacks, i.e. preventing key distribution messages from being dropped due to attacker flooding. As the routing system is a "closed" system, i.e. routers only accept routing messages from known peers, it is possible to configure the routing system to prevent malicious hosts from injecting routing packets. For instance, two adjacent routers may be configured to only accept routing packets from the port connecting to the peer with a TTL equal to 255; non-directly connected routers (such as iBGP peers) may be configured to communicate using MPLS tunnels. Routers can then forward and process routing packets with the highest priority. Normal data traffic cannot congest the routing channel, and therefore cannot DoS the key distribution process. And if the routing channel is congested, a router can determine the misbehaving peer and cut off that peer.

3.3 Prevent Replay Attacks

The packet passport system we described so far prevents forgery, but does not prevent replay attacks. An attacker that intercepts a packet with a valid passport can duplicate the packet multiple times to launch flooding attacks. This may cause a router to block an innocent source whose packets are replayed by attackers.

Our design prevents replay attacks with a combination of rapid rekeying and bloom filters [12]. We discard timestamps because they require global clock synchronization, and we discard sequence numbers because they must be synchronized among border routers of a domain.

Our design uses a hash chain to provide rapid rekeying without excessive key distribution messages. A source domain AS_i does not use the secret key $K(AS_i, AS_j)$ to generate and validate MACs directly. Instead, both AS_i and AS_j use this key to seed a hash chain $\{K_m(AS_i, AS_j) | K_m(AS_i, AS_j) = Hash^m(K(AS_i, AS_j))\}$ that is used in a descreasing order. Each $K_m(AS_i, AS_j)$ is used to generate and validate MACs for a short interval such as a few seconds. When stamping a passport, the source domain attaches m to specify which keys are used. For instance, in the example shown in Figure 1, m being 2 indicates that the keys used to generate the MACs are $\{K_2(AS_1, AS_j), j = 2, 3, 4\}$. A transit or destination domain uses the index m in the passport to locate the proper key and validates the passport. A lower index $m - 1$ invalidates all keys with a higher index, thereby preventing replay attacks. A transit or destination domain AS_j may keep a window of two keys $K_{m-1}(AS_i, AS_j)$ and $K_m(AS_i, AS_j)$ to deal with out of order packet delivery or out of sync key indexes, i.e. a border router of the source domain AS_i may advance to $K_{m-1}(AS_i, AS_j)$ while other routers of AS_i are still using $K_m(AS_i, AS_j)$.

Domains also need to periodically change the secret keys $\{K(AS_i, AS_j)\}$ for better security. This requires periodic distribution of new $\{d_{AS_i}\}$, which can happen at a large time scale. For instance, if each key $K(AS_i, AS_j)$ lasts for a day, then a hash chain of a length 17280 will allow m to change every five seconds.

Rapid rekeying prevents an attacker from replaying a packet passport a few seconds old. However, an attacker may still replay sniffed traffic on the fly, e.g. duplicating every sniffed packet right after it receives the packet. To prevent this type of attack, we use a bloom filter to track packet passports generated with the same hash key, similar to the technique used in [27]. We add a 64-bit packet ID field in a packet passport. A border router of a source domain will stamp a unique ID into the packet passport and include the ID in its MAC computations. (To ensure uniqueness in the same domain, each border router may start with a different ID space.) When a router in a transit domain receives the packet and validates its passport, it hashes the packet ID and the origin AS number into a bloom filter, and discards the packet if the bloom filter indicates the passport is a duplicate.

A bloom filter has a false positive rate, but this rate can be reduced to a very small value by increasing the filter size. Moreover, it is even possible to accommodate the false positives by allowing a small number of "duplicate" passports to pass. For instance, a router can keep a counter of how many duplicates its bloom filter

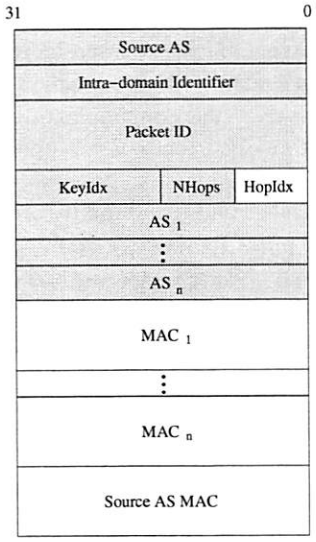

Figure 2: **The format of a packet passport.**

has reported. The counter is reset when the bloom filter is refreshed. If the counter is smaller than a threshold, the router assumes the duplicates are actually false positives and would not discard them; otherwise, the router knows that passport replay attacks are going on and discard any duplicate. The false positive rate of a bloom filter can be computed given its parameters [12], and the threshold can be set such that without attacks all false positives can pass the router. Allowing some duplicate passports to pass gives attackers the possibility of more advanced passport replay attack; the impact of such mechanisms is still under investigation.

3.4 Passport Format

Figure 2 summarizes the format of a packet passport. The source AS field and the intra-domain identifier field together constitutes a source identifier.[3] The packet ID field is used for bloom filter hashing. The KeyIdx is the current hash key index m as described in 3.3. The NHops and HopIdx fields specify the total number of domain-level hops and the current hop. The rest of the passport is the AS path the packet takes, followed by a sequence of MACs. The MACs are computed in a descreasing order, i.e. MAC_n is computed first and MAC_1 is computed last. MAC_j will be validated by AS_j when the packet is forwarded to its destination. MAC_j is computed from the secret key shared between the source AS and AS_j, the source and destination addresses in the IP header, the shaded fields in Figure 2, and all the existing MACs $\{MAC_i, i > j\}$[4]. The last MAC, the source AS MAC, is computed by the source AS using a key known only to itself. This MAC is for the source AS to verify that a passport is generated by itself. We note that if we include more fields and the payload in the packet into the MACs, we can also protect the integrity of the packet. But this requires more computation at packet forwarding time. It's our future work to evaluate the cost and benefit of protecting packet integrity at the network-layer.

Our design uses a 64-bit MAC for each AS hop. The length of the MAC is a tradeoff between security and the packet header overhead. As the hash key used to compute MACs changes rapidly, we think that a 64-bit MAC offers acceptable security.

3.5 Applications

The immediate application of a packet passport is to filter unwanted traffic, because a passport shows the real source of a packet. Suppose host B in Figure 1 classifies packets from host A as attack traffic and wants to block traffic from A. B could send filtering requests to AS_1 as described in [4] to block A. The packet passports stamped by AS_1 would serve as a proof-of-victim, because B cannot possibly forge the passports.

Packet passports may also be used to limit reflector attacks. If each transit domain checks whether the source address of a packet belongs to its origin domain, cross-domain source address spoofing is effectively eliminated and network-layer reflector attacks are limited such that the attackers and the victim must be in the same domain. If a domain can further eliminate source address spoofing inside itself, attackers in this domain cannot perform network-layer reflector attacks at all.

Packet passports may enable new types of resource allocation schemes. For instance, ISPs can queue packets based on their domain identifiers. This scheme would allow ISPs to assign different weights to different domains based on how much they pay. In addition, this also enforces fate sharing among packets from the same domain, and encourages local security. If a compromised host launches DoS attacks, it will only congest the queue for its domain and cause damage to other hosts in the same domain.

3.6 Incremental Deployment

Our design supports incremental deployment and provides incentives for early adoption. Domains can embed their keys in BGP prefix announcements as optional and transitive path attributes. Domains with passport system would extract keys from these attributes, while domains without passport system would pass on these attributes to

[3] The current AS number in the Internet is 16-bit, but IETF is working on the transition to 32-bit AS numbers. Our design assumes a 32-bit AS number.

[4] This way, if a malicious router corrupts MACs for down-stream domains in a packet, the packet would be dropped at the next passsport-checking router and could not consume more bandwidth.

other domains. In case of a partial deployment, the AS path in a passport generated by the source domain only includes the domains that support passports. Passport-enabled domains will validate the passports, and treat legacy packets and packets with invalid passports with low priority. Domains without passport system would ignore the passports and forward the packets.

Early adopters of the passport system can gain the following advantages. First, a domain that deploys the passport system can prevent other domains from spoofing its source identifiers. Second, it can identify any attack source from another domain that also supports the passport system. Third, it can locate any attack source in itself when a victim presents the passports of the attack traffic. Fourth, it can avoid liability for attacks that claim to originate from it but are actually not. Lastly, its packets will treated with higher priority at domains that also support the packet passport system.

3.7 Packet Fragmentation

Unfortunately, packet passports are incompatible with fragmentations at intermediate routers. If an intermediate router fragments a packet, a fragment either has a duplicate passport, which will be caught by the bloom filters as a replay, or does not have a valid passport. Therefore, end hosts should use path MTU discovery to avoid fragmentations. We do not think this incompatibility a significant obstacle to deployment, because fragments are discouraged due to various performance and security issues. IPv6 has already disabled the use of fragmentations at intermediate routers.

4 Feasibility Analysis

It requires a comprehensive experimental study to evaluate the performance and feasibility of our design. As a first step, we provide a preliminary analysis to assess the feasibility of the design. Our design has three primary sources of cost: 1) the computational cost for passport generation and validation; 2) communication and computational cost for key distributions; 3) computational and memory cost introduced by bloom filters.

First, we estimate the computational cost for passport generation and validation. With our design, the source domain of a packet needs to compute L MACs to generate a passport, where L is the length of an AS path. A transit domain or the destination domain needs to compute one MAC to validate a passport. MAC functions such as UMAC [19] can be computed efficiently on modern processors. Speed test with UMAC implementation from [1] shows that a Pentium(R) 4 2.80GHz processor is able to compute 3.9 million HMACs per second over 64-byte blocks. If we assume the average AS path length is four [10], the input used to compute a packet passport is less than 64 bytes. A commodity PC can generate 975K passports and validate 3.9 million passports per second. In a router we expect that special hardware can perform MAC generation at a much higher speed.

Second, we analyze the communication and computational cost for key distributions. With BGP, the key distribution process requires that one public key (probably with certificate) and one digitally signed message with AS_i and d_{AS_i} as the content be appended to each UPDATE message, and d_{AS_i} be refreshed every key distribution interval. With hash chains, the key distribution interval can be relatively long, typically 24 hours, and therefore we think both the communication and computation cost is affordable.

Finally, we look at the memory cost of bloom filters. Our design uses bloom filters to prevent passport replay within the short life span of a hash key, which is on the order of seconds. Bloom filters have a false positive rate $P = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k$, in which k is the number of hash functions, m is the size of the filter in unit of bits, and n is the total number of key values that are already in the filter [12]. When $k = 8$, and the memory efficient factor m/n is 32, we can achieve a false positive rate of 5.7×10^{-6}, which we think is acceptable in the Internet. With these parameters, if we use a filter of 16MB SDRAM and assume an average packet size of 400 bytes [2], a bloom filter can "remember" 2.5Gbps traffic for 5 seconds. For a window of two overlapping keys as described in Section 3.3, two bloom filters with a total memory of 32MB can prevent replay attacks from a 2.5Gbps incoming link.

5 Related Work

Most related work falls into two categories. The first category includes work on preventing source address spoofing, i.e., ingress filtering [13], SAVE [21], route-based filtering [24], and reverse path filtering [22]. In contrast to packet passports, source addresses are not self-verifiable. A destination ISP cannot trust a source address unless it assumes the rest of the network has eliminated source address spoofing.

There is also work on detecting source address spoofing near the destination, such as Hop-count Filtering [16]. The routers near the destination can simply drop packets with spoofed source addresses. However, this type of approaches cannot identify the attacking source and can only protect the last-hop link.

Another category of work focuses on providing unspoofable identifiers with cryptographic primitives. Perlman's work on sabotage-proof routing uses heavy-weight public-key digital signatures [26], while our work uses light-weight MACs. PI [30] and AITF [4] use a path identifier to approximate an unspoofable source identifier. Routers insert secure tokens into a packet before

they forward the packet. Although PI and AITF have lower packet header overhead, a notable challenge for them is that a source can prepend a path identifier with arbitrary values, and a compromised router can also forge the path identifiers. In contrast, packet passports are cryptographically unforgeable. Source identifiers protected by packet passports can be directly used in filter expressions to block attack sources.

Authenticated Marking Scheme [28] also uses MACs to protect router-inserted path identifiers, but the keys are revealed to verifiers after generating MACs. A verifier has to store received packets for a period of time before being able to verify the MACs. In contrast, packet passports can be verified on the fly, and do not require additional protocol messages to reveal MAC keys.

SPM [3] uses secrets shared between domains as proof-of-source, but the secrets are put into packet headers as plain text and therefore vulnerable to eavesdroppers. Moreover, the secret distribution of SPM is not DoS resilient: if links used to distribute the secrets are flooded, the secret distribution itself would fail. Our packet passport system can accommodate eavesdroppers, and our key distribution is fully protected against DoS attacks.

The Visa protocol [11] shares similarity with our work. A packet carries an exit visa to leave its source domain, and an entry visa to enter a destination domain. The Visa protocol requires per-connection Visa request, but our work does not. The visa request channel still needs to be protected from DoS attacks and address spoofing. The Visa protocol can use our passport system to protect its request channel. In a similar vein, capability-based systems such as SIFF [31] and TVA [32] and ticket system [25] can also use packet passports to protect their request channels.

6 Conclusion

This paper proposes a packet passport system that securely and efficiently authenticate the source of a packet. A packet passport is crytographically unforgeable. Routers at passport checking points can independently validate the authenticity of a packet passport without trusting the rest of the network. Destinations can use the passports from attack packets to accuse the attack sources, and routers can use them to validate source identities and block attack sources.

We present the design of a packet passport system that uses a light-weight MAC, introduces low message overhead, and is incrementally deployable. A preliminary study suggests that the design is feasible with today's hardware technology.

References

[1] UMAC implementation. `http://www.cs.ucdavis.edu/~rogaway/umac/`.

[2] Caida report. `http://www.caida.org/analysis/AIX/plen_hist/`, February 2000.

[3] B.-B. A. and L. H. Spoofing prevention method. In *Proc. IEEE Infocom*, 2005.

[4] K. Argyraki and D. Cheriton. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *USENIX 2005*, 2005.

[5] DDoS attacks still pose threat to Internet. BizReport, 11/4/03.

[6] D. Boneh and M. Franklin. Identity based encryption from the weil pairing. *SIAM J. of Computing*, 32(3):586–615, 2003.

[7] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, and A. S. andJacobus van der Merwe. Design and implementation of a routing control platform. In *Proc. of NSDI*, 2005.

[8] Extortion via DDoS on the rise. Network World, 5/16/05.

[9] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[10] D.Magoni and J.J.Pansiot. Analysis of the autonomous system network topology. *Computer Communications Review*, 31(3):26–37, July 2001.

[11] D. Estrin, J. C. Mogul, G. Tsudik., and K.Anand. Visa protocols for controlling inter-organization datagram flow. *IEEE Journal on Selected Areas in Communication*, 7(4):486–498, May 1989.

[12] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. Technical Report 1361, Department of Computer Science, University of Wisconsin-Madison, 1998.

[13] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks that Employ IP Source Address Spoofing. Internet RFC 2827, 2000.

[14] S. Gibson. The strange tale of the attacks against grc.com. `http://grc.com/dos/grcdos.htm`, 2005.

[15] A. Greenhalgh, M. Handley, and F. Huici. Using routing and tunneling to combat DoS attacks. In *Proc. of USENIX SRUTI*, 2005.

[16] C. Jin, H. Wang, and K. G. Shin. Hop-count filtering: an effective defense against spoofed ddos traffic. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 30–41, 2003.

[17] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (s-bgp). *IEEE Journal on Selected Areas in Communications*, 2000.

[18] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. Internet RFC 2104, 1997.

[19] T. Krovetz. UMAC: Message authentication code using universal hashing. Internet RFC 4418, 2006.

[20] R. Lemos. A year later, DDoS attacks still a major web threat. `http://news.com.com/A+year+later,+DDoS+attacks+still+a+major+Web+threat/2009-1001_3-252187.html`, Feb. 2001.

[21] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang. SAVE: Source address validity enforcement. In *Proc. of INFOCOM*, 2002.

[22] K. Miller. Three practical ways to improve your network. In *Proc. of USENIX LISA*, 2003.

[23] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *Usenix Security Symposium*, Aug. 2001.

[24] K. Park and H. Lee. On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets. In *ACM SIGCOMM*, 2001.

[25] B. Patel and J. Crowcroft. Ticket based service access for the mobile user. In *Proceedings of MobiCom '97*, pages 223–233, 1997.

[26] R. Perlman. Network Layer Protocols with Byzantine Robustness. MIT Ph.D. Thesis, 1988.

[27] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer. Hash-Based IP Traceback. In *ACM SIGCOMM*, 2001.

[28] D. Song and A. Perrig. Advance and Authenticated Marking Schemes for IP Traceback. In *Proc. IEEE Infocom*, 2001.

[29] R. White. Securing bgp through secure origin bgp. *The Internet Protocol Journal*, 2003.

[30] A. Yaar, A. Perrig, and D. Song. Pi: A Path Identification Mechanism to Defend Against DDoS Attacks. In *IEEE Symposium on Security and Privacy*, 2003.

[31] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *IEEE Symposium on Security and Privacy*, 2004.

[32] X. Yang, D. Wetherall, and T. Anderson. A DoS-Limiting Network Architecture. In *ACM SIGCOMM*, Philadelphia, PA, Aug. 2005.

[33] H. Yu, J. Rexford, and E. Felten. A distributed reputation approach to cooperative internet routing protection. In *Proceedings of Workshop on Secure Network Protocols*, 2005.

Cookies Along Trust-Boundaries (CAT): Accurate and Deployable Flood Protection

Martin Casado Aditya Akella Pei Cao Niels Provos Scott Shenker
{casado,cao}@cs.stanford.edu,aditya@cs.cmu.edu
niels@google.com, shenker@icsi.berkeley.edu

Abstract

Packet floods targeting a victim's incoming bandwidth are notoriously difficult to defend against. While a number of solutions have been proposed, such as network capabilities, third-party traffic scrubbing, and overlay-based protection, most suffer from drawbacks that limit their applicability in practice.

We propose CAT, a new network-based flood protection scheme. In CAT, all flows must perform a three-way handshake with an in-network element to obtain permission to send data. The three-way handshake dissuades source spoofing and establishes a unique handle for the flow, which can then be used for revocation by the receiver. CAT offers the protection qualities of network capabilities, and yet does not require major architectural changes.

1 Background and Motivation

Denial-of-service (DOS) via packet flooding remains a serious problem on the Internet today. Receiver centric solutions are generally ineffective against flooding because the resource being exhausted is not under the victim's control. Other forms of resource consumption DOS, such as outgoing bandwidth exhaustion or computational complexity attacks, can be treated as a local resource management issue and are often dealt with at the victim with resource scheduling [21] or admission control [15] schemes.

Industry and the Internet research community have proposed or deployed a number of solutions which have made great strides towards offering protection from flooding. We describe some of them here:

Traffic filtering near the victim: ISPs use a variety of statistical and machine learning techniques to "infer" attack traffic with little input from victims, e.g. Push-Back [16], and Arbor Networks [1]. The location where the filtering is performed is either at the tier-1 ISP [1], pushed to ISPs close to the sources [16], or pushed to points upstream of the local bottleneck link of the victim [13]. In general, these schemes provide a trade-off between the rate of false positives and the degree of sensitivity to low-bandwidth attacks.

Traffic filtering near the source: Other schemes suggest filtering packet floods as close to the source as possible, e.g. AITF [7] and Firebreak [11]. However, we believe such schemes are unlikely to be adopted because they involve asking an ISP to take damaging action against one of their customers on behalf of a victim that they have no economic or trust relationship with[1].

Overlays: Overlay solutions, such as CDNs, Mayday [6] and SOS [14], filter packet floods at a large set of nodes distributed across the edges of the network. The aggregate bandwidth offered by the overlay nodes can be very high. However, even in the presence of a high-bandwidth overlay to vet traffic, the back-end server's public IP address remains vulnerable.

Network Capabilities: In a departure from filtering-based approaches, network capability schemes, such as SIFF [22] and TVA [23], require recipients to mark wanted/legitimate traffic with unforgeable capabilities. These schemes guarantee that legitimate traffic always receives priority over illegitimate traffic. However, existing proposals require changes to both clients and servers, as well as upgrades to routers to honor capabilities and protect legitimate traffic. As a result, they do not provide tangible near-term benefits.

Third party traffic scrubbing: In this approach, a company peers with a number of large ISPs, and sinks all traffic to the protected servers. The company can then offer DoS prevention services to the web-servers such as protection from connection floods (by performing the TCP handshake in the network), flood detection (using statistical techniques mentioned above) and per-flow rate limiting (e.g. Prolexic [3]). However, with this approach it is difficult for web sites to control decisions regarding the legitimacy of traffic. Generally, the company offering protection handles both the detection of malicious traffic

[1] [13] provides a good argument in support of this position

and the enforcement.

In this paper, we propose CAT, a new approach for protecting public web servers against bandwidth floods. Our goal for CAT is to offer the same protection guarantees as capabilities while having a practical adoption story. This implies no major architectural changes, backwards compatibility with existing clients and an incremental deployment path. CAT achieves this goal through combinations of the following techniques:

- *flow cookies*: This is a stateless, backward-compatible capability mechanism that offloads connection-setup and filtering to in-network elements without requiring per-flow state.

- *filtering on trust boundaries*: Existing commercial relationships among ISPs can be leveraged to identify viable locations for filtering. We term the furthest set of ISPs with which a web server has an economic relationship its *trust boundary* and argue that the trust boundary is a natural location for filtering.

All flows to a CAT protected server must first perform a handshake with a middlebox on the server's trust boundary. Successful completion of the handshake will provide the sender with a valid flow cookie which will allow the sender to communicate with the receiver for the lifetime of the cookie or until the receiver revokes it. Cookie revocations are performed by the middlebox on the trust boundary.

In the following sections we present an overview of CAT (§2), then discuss flow cookies in more detail (§3) and a mechanism for dynamically detecting the trust boundary (§4). In §5 we present an analysis of BGP data to determine the properties of existing trust boundaries. Finally, we present related work in §6 and conclude in §7.

2 Overview of CAT

CAT was designed around two guiding principles:

1. For filtering mechanisms to be accurate and effective, decisions about whether or not to permit flows must be made by devices or services that can reliably determine both the *origin* of traffic as well as whether the destination *wishes to receive* the traffic.

2. To ensure incremental deployability of a protection mechanism, existing commercial relationships between network entities (e.g. ISPs) can be leveraged as forcing functions for the deployment of protection infrastructure.

The *flow cookie* mechanism realizes the first principle. Flow cookies liberally borrows ideas from capabilities and third-party traffic scrubbing. Like third-party scrubbing, a "cookie box" is situated on-route between the clients and protected web-server at a network location with high incoming bandwidth (typically inside the protected server's ISP). The cookie box engages all flows from clients in a three-way handshake. Only connections that have completed the handshake are forwarded on to the web server. Subsequent to the handshake, flow cookies functions similar to network capabilities. The cookie box inserts capabilities in outgoing packets from the server, and these capabilities must be echoed in future packets from clients. We use the TCP timestamp field to insert the capability, thereby ensuring backwards-compatibility. When the web-server deems a client to be misbehaving, it can terminate the connection and/or simply request the cookie box to filter the offending flow or IP.

This simple approach offers several salient features:

- Offloading the three-way handshake to high-bandwidth infrastructure allows special purpose hardware to determine if clients are source spoofing.

- All packets forwarded to the server must carry a non-forgeable cookie which is computed over the packet header. Therefore, web-servers can make effective policy decisions on the basis of IP addresses which are enforced in the network.

- The signaling of the "wanted-ness" of a flow is implicit via response traffic from the protected server. That is, if a server does not reply to a client with an ACK, the client will be unable to get a fresh cookie after its current one times out.

- With filtering enforced in-network, this approach offers web servers the protection bandwidth of the link(s) over which the cookie box is deployed.

- Detection of illegitimate traffic is performed at the web server, hence the server can implement flexible and meaningful policies for its particular service.

As described above, flow cookies operates through a point deployment, and is therefore easy to adopt. However it suffers from two key limitations: (1) if an attacker overwhelms the link on which the cookie box is deployed, she can effectively flood the server (2) unless the cookie box sinks all traffic, packet floods can avoid filtering by routing around the cookie box (e.g. using alternate ISPs or peering points).

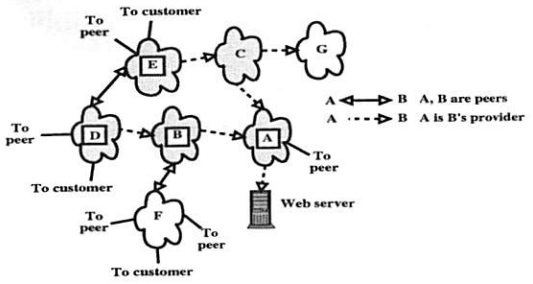

Figure 1: ISPs shaded gray are in the region of trust of the web server. ISPs enclosed in a white square are on the trust boundary For example, ISPs A and B are on the trust boundary since they terminate the trust chain for traffic received from their peers.

Naturally, these limitations can be addressed by deploying more cookie boxes at a diverse collection of network links spread across several ISPs. We note that deployment of cookie boxes can be facilitated by the web server taking advantage of the network links for which it, or its ISP, or recursively its ISP's ISPs, have an *economic relationship* (Two network entities have an economic relationship if one *pays* the other for delivering traffic). The transitive closure of such economic relationships among ISPs defines a "region of trust" for each protected server (see Figure 1). We consider ISPs in the trust region that terminate a sequence of economic relationships to be on the *trust boundary* (Figure 1).

Links on a server's the trust boundary are natural locations for deploying cookie boxes because they offer the highest incoming bandwidth without requiring the web-server to forge new relationships. We argue that not only does a server have stronger clout to persuade an ISP within its region of trust to deploy a cookie box, it also has greater leverage to ensure that the cookie boxes will be administered and operated correctly

To be effective, a trust boundary must have the following two properties. First, the aggregate bandwidth across all boundary links must be very high. This ensures that cookie boxes on the trust boundary can handle high-volume flooding. Second, the number of routes traversing the boundary links should be roughly proportional to the bandwidth of the links. This prevents in-network hot-spots from underprovisioned links managing too many flows while other links on the boundary have available filtering capacity. In §7, we present an analysis of AS level connectivity graphs for the Internet which demonstrates that the trust boundary for most stub networks today satisfy both of these properties.

By definition, ISPs on the trust boundary of a given web server must perform the TCP handshakes and subsequent filtering on all traffic to the web server. To effectively provide this functionality in a setting where ISPs could lie on one of several trust boundaries (and inside several other regions of trust), the ISPs must coordinate among each other to decide who must perform the TCP handshake and subsequent filtering for a particular web server. In §4 we present a simple modification to BGP by which multiple ISPs can ascertain their responsibility in a distributed manner.

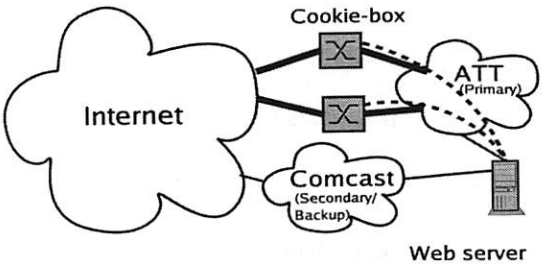

Figure 2: A multihomed web server seeking protection deploys cookie boxes at some of the network links of its primary ISP. These could be edges along which the web server expects to receive a large fraction of its traffic.

3 Flow Cookies

Flow cookies is an extension of SYN cookies[8] wherein a middlebox places a secure, limited lifetime cookie within the TCP timestamp of every outgoing data packet from the protected server. Flow cookies offers strong protection against flooding, does not require modification to clients or to the network, is resistant to source-spoofing, and does not require per-flow state in the network.

Unlike approaches common in the commercial world, flow cookies is not meant to be deployed at a web server's perimeter. Instead, a "cookie box" (or a handful of boxes) which enforces flow cookies are deployed at the edges of the protected web server's ISP. We illustrate an idealized setup in Figure 2.

In this section, we present a brief overview of the flow cookies approach. Flow cookies was designed to protect services using TCP. We assume that other protocols, such as UDP, are managed through other means, such as rate limiting aggregates. The full details of the flow cookie protocol, design issues and analysis may be found in [9].

3.1 Overview

The cookie box(es) and the protected web server cooperate to perform the following four tasks:

1. The cookie box intercepts all SYN packets destined to the web server. If the SYN packet's source IP is in an "`IP blacklist`", it is dropped (i.e., the IP blacklist is only looked up for SYN packets). Otherwise, the box responds with a SYN cookie [8] in which the source address is forged to be that of the web server. The cookie is computed using a keyed message authentication code

over the connection 4-tuple. SYN cookies does not require state maintenance at the cookie box and can be run at gigabit line speeds [2].

This first step ensures that spoofed sources, SYN floods and unwanted connection attempts cannot traverse the link between the cookie box and the protected website.

2. For packets to the web server that carry an ACK flag (this includes all data packets), the cookie box checks that the ACK'ed sequence number is a valid SYN cookie. If it is, the connection is handed off to the web site using a TCP handoff scheme such as [4].

3. For outgoing packets from the web server to its clients (this happens after the web server has accepted the client connection request), the cookie box adds a secure non-forgeable "flow cookie" (explained below).[2] The flow cookie is computed in a similar manner as the SYN cookie and is valid for a limited amount of time (e.g. 1 minute). As we explain below, the flow cookie is echoed back by the client, and checked by the cookie box.

The cookie box implicitly infers whether the web server wishes to engage in communication with a particular end-point by marking outgoing ACKs. If the web server chooses not to send ACKs to a client, the client will be unable to get fresh cookies once the old one times out. The flow cookie further helps the cookie box ensure that only packets belonging to flows accepted by the server traverse the link between the cookie box and the web server.

4. The web server understands the accepted usage policy of its local administration, and is already keeping per-flow state for each outstanding connection. Therefore, it is in the best position to determine if a client is misbehaving.

If the web server does not want to receive packets from a particular flow, it can do two things: (1) push filters to the cookie box's "`flow blacklist`"; in this case, the cookie box maintains the source IP and port in a revocation list with an associated time out and filters packets accordingly. Or, (2) inform the cookie box to stop issuing fresh capabilities for the client. This can be done statelessly by simply closing the connection, in which case the client will no longer receive valid cookies for the flow (after the current cookie times out). The first approach can be employed to filter high-bandwidth malicious flows immediately. The second approach can be used in less critical situations or to shut off low priority clients when under overload.

If a server determines that an attacker is behind a given IP address (or address block), it can request the cookie box to add the IP (address block) to the `IP blacklist`.

[2] We assume that Internet routes are symmetric at the AS-level.

3.2 Ensuring Backwards-Compatibility

To be backwards-compatible we exploit the *TCP timestamp* option and place the flow cookie from step #3 in the timestamp field of packets.

The TCP timestamp option, proposed in RFC-1323 to measure RTTs, is supported by all the major host operating systems. According to the RFC, once the option is enabled by both ends, the sender places a timestamp in a packet, and subsequent packets from the receiver *echo* the timestamp. Common operating systems enable timestamps by default, with the exceptions of Windows2000 and WindowsXP. As has been shown by [20], it is possible to trick Windows into echoing timestamps by including a timestamp option in the SYN ACK packet.

On connection set up, the cookie box negotiates the timestamp option with the client. RFC-1323 does not specify how the timestamp value is to be encoded in the option field. To prevent disruption, the cookie box and the web server must explicitly agree on a format. Also, care needs to be taken if the web site wishes to use timestamps to measure RTTs. To avoid undesirable interactions, protected web servers could be dissuaded from using timestamp values in RTT calculations. This requires a relatively insignificant modification to the TCP stack at the web server. Also, all network stacks are able to do RTT calculations without the aid of timestamps. An alternate method that does not require any modification of the web server is described in full detail in [9].

3.3 Implementation

To demonstrate flow cookies' ability to inter-operate with existing clients, we implemented flow cookies within a software router using a user-space network development environment [10]. We tested our implementation using numerous public web servers. Our tests show that flow cookies is compatible against various client OSes and server software and can handle fully dynamic, interactive sessions as well as static content.

We also found that the flow cookie implementation and IP blacklist lookup had little effect on the throughput of our router and was able to operate at gigabit speeds during micro-benchmarking. While our focus was on a software implementation, we believe that flow cookies can be easily implemented in hardware.

We also tested against synthetic flooding attacks and found that flow cookies is indeed able to offer full protection of established flows even under severe loads. Full details of our implementation and testing are available in [9].

4 Leveraging Trust Relationships

A primary limitation of flow cookies as described thus far is that it only leverages the protection bandwidth of a few high-speed links. So either the protected links must sink all traffic to the web server (this may require changes to routes and could introduce network bottlenecks) or the server is vulnerable to attacks on unprotected links. A more troublesome problem is that, when several cookie-boxes are deployed across many ISP—where each cookie box could be protecting a distinct set of web servers—the cookie boxes must arrive at a consensus on which of them should perform the handshake and filtering for a particular flow.

In this section we extend flow cookies from being a point-solution to a more generic wide-area service. We introduce a minor modification to BGP that allows ISPs which have deployed cookies-boxes to determine which boxes manage the handshake and do the filtering on a per destination basis. We also argue that the most viable and effective deployment strategy is to leverage existing client/provider relationships.

4.1 Trust Relationships Between ISPs

In the Internet today, neighboring ISPs rely on contractual agreements or SLAs to coordinate mutual exchange of traffic. While some agreements involve money changing hands (such as customer-provider contracts), others are more faith-based (e.g. peering relationships) [17].

We argue next that such agreements also define "trust relationships" between ISPs, either implicitly or explicitly. By a trust relationship between network A and network B, we mean that A trusts B to take the necessary steps from preventing packet floods from entering into A's network via the A–B link.

In the simplest case, if ISP A pays ISP B for global Internet connectivity, then it is safe to say that "A implicitly trusts B"; a violation of this trust – detectable at A, for example, when B lets packet floods through – will likely result in A picking an alternate provider. Such implicit trust relationships already exist today.

Neighboring ISPs, that don't share a customer-provider relationship, may also enter into *explicit* trust relationships. For example, two peer ISPs may contractually agree to mutually cooperate and vet packets destined for each other's network. Several such instances of cooperation among peer ISPs for traffic engineering purposes have been observed empirically. Explicit trust relationships may also be forged between a customer network and a non-neighbor ISP providing a specialized form of protection.

By default, trust relationships are not symmetric. For example, a customer may trust its provider to do the right thing, but the provider cannot assume any guarantees from its customer. However, most explicit trust agreements between peers are likely to be symmetric.

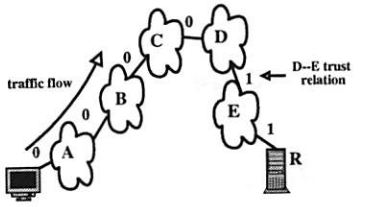

Figure 3: An illustration of trust relationships on an end-to-end AS-level path.

In what follows, we assume trust exists in the customer-provider relationships, not in peering relationships.

4.2 A BGP-based Mechanism

Our insight is that we can leverage BGP and existing trust relationships between ISPs to determine, in a completely distributed and stateless manner, a single ISP that is responsible for inserting flow cookies and filtering packets.

We illustrate our solution through an example. Say source S wants to send to receiver R. By the design of routing in the Internet, S knows the "AS-level path vector" to communicate with R via BGP: `A, B, C, D, E` (Figure 3). Here A is S's ISP and E is R's ISP.

We assume that every ISP along the path vector knows the relationships between pairs of ISPs downstream from S towards R. For example, A knows the trust relationship for `A--B`, `B--C`, `C--D`, `D--E`, and `E--R`. B knows `B--C`, `C--D`, `D--E`, and `E--R` and so on.

This information is easy to piggy back on to the BGP path vector protocol: In BGP, when an AS wants to announce a route to a neighbor, the AS appends its AS number to the announcement. We simply change the announcement to also indicate whether the AS trusts the neighbor. The destination prefixes to be protected are similarly advertised in the BGP announcement.

Lets says that the trust relationship between a pair of neighboring ISPs, e.g., A-B, is encoded in a binary form: 1 means that B trusts that packets received from A have been vetted by A using flow cookies, [3] and 0 means otherwise.

Assume the trust relations on the path from S to R are as follows:

```
S--A: 0
A--B: 0
B--C: 0
C--D: 0
D--E: 1
E--R: 1
```

[3] a 1 also implicitly assumes that A has flow cookies deployed

If a packet from S to R arrives at A, A checks notices that the sequence of trust relationships along the forward path to R contains at least one "0". A simply forwards the packet along. Similarly, B and C forward the packet along.

When D receives the packet, it notices that there is a continuous trust sequence of 1's, starting from D, all the way to the receiver R; and that D did not trust its upstream neighbor, C. D would also check that the destination prefix, R, was advertised to be protected.

For ISPs such as D to make this inference, we must make minor modifications to the routing tables at their cookie boxes. Assume that the ISP D deploys a distinct cookie box on each of its peering links. The cookie box is similar to a router, with its own routing table etc. The only difference is that each route in the cookie box's table is annotated with the "AND" of the downstream trust sequence, and the inverse of the trust relationship with the upstream. For example, annotation at D for packets from received from ISP C destined for R is:

$(T_{RE}$ AND $T_{ED})$ AND (NOT$(T_{CD}))$

Where T_{XY} denotes the trust relationship between X and Y. If the annotation is 1, then D does the flow cookie check, insertion or filtering (see below). Otherwise, D lets the packet through. Based on this inference, in the above example, ISP D decides that it must do the flow cookie insertion/check for packets from S to R.

As packets flow past the trust frontier, they are forwarded to the destination. For example, E knows that D can be trusted to vet packets. E simply forwards the packet along. Similarly, R trusts E and accepts the packet as legitimate.

Trust relationships similarly come into play when deciding to filter packets. When R wants to filter packets, it pushes filters upstream to E. E in turn pushes to D, and this happens until the filters hit a "trust boundary" (e.g. the edge between D and C), when the filters are finally installed.

As filters are pushed up the trust chain, tier-1 ISPs may have to install a large number of filters. However, as we described in Section 3.1, filters are mainly needed for connection attempts; attack flows are automatically filtered when the timer for their cookie expires. Therefore, filters can be stored in larger, slower memories since they are only consulted on SYN packets. Further, the IPs can be aggregated in the filters under memory pressure.

4.3 Exploring the Trust Boundary

As described previously, the trust relationships between ISPs implicitly define a region of trust within which a protected server could deploy multiple cookie boxes. We refer to the boundary of this region as the "trust boundary". For example, ISP D above is on the trust boundary

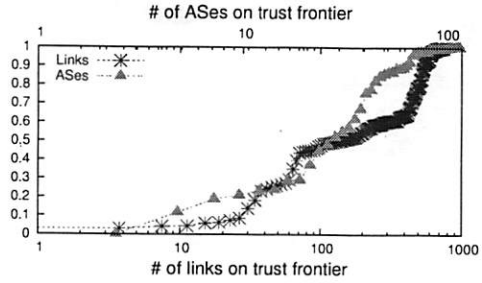

Figure 4: CDF of the number of ASes (top x-axis) and links (bottom x-axis) on the trust boundaries of 13,031 stub networks.

of stub network R for all packets arriving from S. A different ISP may be on the trust boundary for packets from other sources.

As past studies have shown, ISPs higher up in the Internet ISP-hierarchy have higher bandwidth links [5]. Hence, for example, links in ISP D above have higher capacity than those in E. This implies that, independent of the mechanism used, the trust boundary dictates the *maximum* amount of aggregate filtering bandwidth available to a server under existing trust relationships. In addition, ISPs higher up in the hierarchy are known to have a much larger number of interconnections with other ISPs [12]. Therefore, the advantage of using ISPs in the trust boundary over a point deployment for filtering is not only the access to greater bandwidth but isolation between multiple links.

Put another way, the ideal trust boundary should be composed of multiple, large ISPs (tier-1 and tier-2) with routes distributed across boundary links in a manner proportional to the link bandwidth.

In what follows, we explore the properties of trust boundaries on the Internet today using publicly available AS provider-subscriber and peering relationship information inferred from BGP data [19]. Our analysis assumes that trust only transfers across client/provider relationships and ends at peering links. In practice, peering agreements may contain provisions for filtering as well, therefore we consider our analysis pessimistic in the amount of filtering bandwidth available on the trust boundary.

The analysis covers the trust boundaries for 13,031 stub networks. We first look at the total number of distinct ASes and links on the trust boundaries for each of the stub networks (Figure 4). Here, and in the subsequent analysis, we use the term "link" to refer to the collection of several physical interdomain links between neighboring ISPs.

Over 70% of stub networks have 10 or more different ASes on the their boundaries, with over 60 total links. 90% of stubs have 20 or more links on their boundary.

We also look at the distribution of tiers on the trust

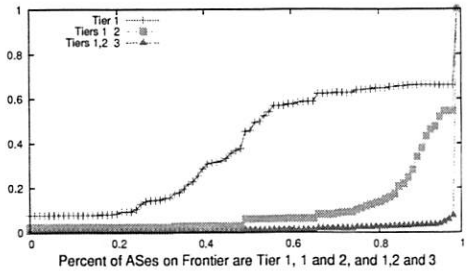

Figure 5: CDF of the percent of ASes on the boundary that are tier 1, tiers 1 and 2 and tiers 1,2, and 3

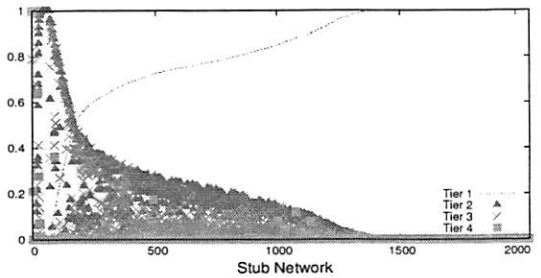

Figure 6: Histogram of the fraction of routes passing through boundary ISPs belonging to tiers 1–4.

boundary. As shown by past studies [5], we assume that links from smaller, or lower-tier ISPs (e.g. tier 3 and 4) are lower bandwidth than those from tier 1 and 2. Figure 5 is a CDF plot of the percentage of autonomous systems that are either tier 1, tiers 1 and 2, and tiers 1,2 and 3. It is clear that the boundary is largely made up of tier 1 and 2 ISPs. The trust boundaries of over 80% of stub networks consist of 85% or more tier 1 or 2 ISPs. This is unsurprising given that larger tier ISPs peer heavily thus ending the recursive passing of trust.

Regardless of the distribution of tiers on the boundary, if many routes to a web servers traverse small ISPs on the trust boundary, then the boundary may still only provide nominal protection. To demonstrate that this is not the case, we analyze routes from all destinations in our data set to each of several stub networks. Figure 6 is a histogram of the percent of routes which traverse each tier (1-4) on the trust boundary for over 8,000 stub networks. As expected, for most trust boundaries, the vast majority of routes pass through tier 1 links. Very few pass through tier 4.

These results suggest that, in most cases, the trust boundary can be used to provide effective, high-speed filtering bandwidth in the network. However, given the nature of peering relationships it is difficult to do a complete study of the trust boundary properties. For example, smaller tier-3 or tier-4 ISPs often engage in private peering relationships. Such relationships cannot be inferred from public BGP data. Another issue that we neglect in our analysis is that the AS level path vector may not necessarily reflect the true forwarding path of a packet (due to internal ISP policies). However, we do believe that our BGP-based scheme and analysis of the trust boundary hold for such situations as well.

5 Related Work

We discussed several related approaches in Section 1. In this section, we elaborate on some of them.

Filter-based approaches, such as Pushback [16] and AITF [7] require the identification and blocking of illegitimate traffic from within the network. In Pushback, a router attempts to identify attack traffic by determining that it is in a congested state, finding an "aggregate" that describes the attack traffic, and pushing the "aggregate" upstream to be blocked close to the source. In contrast, our solution lets the web server discern unwanted traffic.

AITF is based on the observation that a pure filter-based approach such as Pushback requires too much state in core routers. AITF decentralizes the state by pushing filtering rules as close to the source as possible. However, it does not consider how the filtering rules are determined and whether the attack recognition mechanism is resistant against spoofing.

Flow cookies can be viewed as a simplified variant of network **capabilities** [22, 23]. In contrast with capabilities, our scheme offloads connection negotiation to the network and requires no modification to routers, nor to IP or TCP. Also, flow cookies aims for "partial path protection", and trusts one end of the communication: the public web sites; Capability schemes strive to offer "complete path protection".

Firebreak [11] proposes to use IP anycast to redirect traffic to filtering portals close to the source. Flow cookies could be used in conjunction with firebreak to enforce precise, per-flow filtering in a stateless and backwards compatible manner. However, we feel that a more viable deployment strategy is to "push" filtering up from the target rather than expect deployment at the edge, close to the source.

In [18], the authors suggest using virtual nets within the network to filter DDoS. Like our proposal, they suggest deployment close to the victim. Authors in [13] extend this to work over multiple ISPs. However, because the filtering points do not participate in flow negotiation, these schemes are vulnerable to source spoofing and first-packet flooding. Also, we offer a method for dynamically determining where to enforce filtering given a placement of the filtering hardware.

DDoS solutions relying on the deployment of security appliances, such as [2, 3], perform a number of DDoS prevention functions such as offloading the three-way handshake with SYN cookies, enforcing per-flow rate limiting of millions of flows etc. We argue that the end-server must be responsible for distinguishing good vs bad

traffic and pushing the filtering decisions to the network. In addition, we do not require per-flow state lowering the complexity and cost of implementation.

6 Summary

DDoS flooding attacks that target the victim's incoming bandwidth are particularly difficult to contain since the victim cannot directly control the utilization of its incoming link. Several solutions have been proposed both in the industry and research community to tackle this problem. However, these solutions are either inaccurate, ineffective against low-bandwidth floods, impossible to incrementally deploy or require expensive state maintenance.

In this paper, we proposed a simple approach for flood protection that directly addresses the drawbacks of existing mechanism. Our approach builds on current signaling mechanisms (TCP's three-way handshake), past work on network capabilities, and Internet trust relationships (ISP customer-provider relationships).

We outlined flow cookies, a simple, backwards-compatible, point-deployable, network-level protection mechanism that can offer high protection bandwidth to public web servers. Flow cookies helps public web servers implement flexible traffic policies at high bandwidth network locations with minimal additional support from the network.

We argued that the protection offered by flow cookies can be enhanced by deploying flow cookie boxes among ISPs that have a direct or indirect economic relationship with the web server. Such ISPs are said to lie in the web-server's region of trust. We presented a simple BGP-based mechanism to coordinate handshake and filtering activities among multiple such boxes. We study several interesting properties of trust regions in the Internet today.

References

[1] Arbor home page. http://www.arbornetworks.com/.

[2] Netscaler syn flood protection. http://www.netscaler.com/docs/library/ NetScaler-SYNDefense.pdf.

[3] Prolexic home page. http://prolexic,com.

[4] Tcpha home page. http://dragon.linux-vs.org/ dragonfly/.

[5] A. Akella, S. Seshan, and A. Shaikh. An Empirical Evaluation of Wide-Area Internet Bottlenecks. In *Internet Measurement Conference*, Miami, FL, Nov. 2003.

[6] D. Andersen. Mayday: Distributed filtering for internet services. In *USITS, Seattle, WA, 2003.*, 2003.

[7] K. Argyraki and D. R. Cheriton. Active internet traffic filtering: Real-time response to denial-of-service attacks. In *USENIX Annual Technical Conference*, 2005.

[8] D. Bernstein. Syn cookies. http://cr.yp.to/syncookies.html, 1996.

[9] M. Casado, P. Cao, A. Akella, and N. Provos. Flow-Cookies: Using Bandwidth Amplification to Defend Against DDoS Flooding Attacks. Stanford HPNG Technical Report, 2006.

[10] M. Casado and N. McKeown. The Virtual Network System. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, 2005.

[11] P. Franics. Firebreak: An ip perimeter defense architecture. http://www.cs.cornell.edu/People/francis/firebreak/hotnets-firebreak-v7.pdf.

[12] L. Gao. On inferring autonomous system relationships in the Internet. 9(6), Dec. 2001.

[13] A. Greenhalgh, M. Handley, and F. Huici. Using routing and tunneling to combat dos attacks. In *Proc. Usenix workshop on Steps to Reducing Unwanted Traffic on the Internet*, July 2005.

[14] A. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services. In *Proceedings ofACM SIGCOMM'02*, 2002.

[15] S. Lee, J. Lui, and D. Yau. Admission control and dynamic adaptation for a proportional-delay diffserv-enabled web server. In *SIGMETRICS '02*, pages 172–182, New York, NY, USA, 2002. ACM Press.

[16] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev.*, 32(3):62–73, 2002.

[17] W. B. Norton. Internet service providers and peering. In *Proceedings of NANOG 19*, Albuquerque, New Mexico, June 2000.

[18] R. Stone. Centertrack: An ip overlay network for tracking dos floods. In *In Proceedings of the Ninth USENIX Security Symposium*, August 2000.

[19] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the Internet Hierarchy from Multiple Vantage Points. June 2002.

[20] A. B. Tadayoshi Kohno and K. Claffy. Remote physical device fingerprinting. In *IEEE Symposium on Security and Privacy*, 2005.

[21] M. Welsh and D. Culler. Adaptive overload control for busy internet servers. In *USITS*, 2003.

[22] A. Yaar, A. Perrig, and D. Song. Siff: A stateless internet flow filter to mitigate ddos flooding attacks. In *In Proceedings of the IEEE Security and Privacy Symposium*, 2004.

[23] X. Yang, D. Wetherall, and T. Anderson. A dos-limiting network architecture. In *Proc. ACM SIGCOMM*, 2005.

Separating Wheat from the Chaff: A Deployable Approach to Counter Spam

Youngsang Shin, Minaxi Gupta, Rob Henderson
Computer Science Dept
Indiana University
Bloomington, IN, U.S.A.
{*shiny,minaxi,robh*}*@cs.indiana.edu*

Aaron Emigh
Radix Labs
Incline Village, NV, U.S.A.
aaron@radixlabs.com

Abstract

Unsolicited bulk email, aka spam is a persistent threat to the usefulness of the Internet. The fight against spam today relies solely on filtering at the recipient's mail server, which can delay mail delivery. We present a spam countering approach consisting of two complementary techniques. The first, *token-based authentication*, can identify emails from valid senders that a user expects to hear from. This identification prevents much of the good mail from being subjected to the filtering process. The second technique, *history-based prioritization*, is designed for the rest of the email. It utilizes past information about the sending mail servers and their domains to prioritize email filtration, thus reducing delays in the delivery of good email over spam. The proposed techniques do not rely on any infrastructure, deployed or otherwise, and any mail server can choose to deploy them independent of the choice made by the any other mail server.

1 Introduction

Some estimates claim that $60 - 80\%$ of emails today are spam. Spam has become such a huge problem that it threatens to render email itself unusable. Estimates on the cost of spam vary but they all are staggering. FTC estimates that an average person spends approximately 10 minutes each day dealing with spam. In fact, at the rate spam is growing the Radicati group estimates that it would cost businesses $198 billion by 2007! Further, an average spam is about 10KBytes in size. This translates into a huge amount of bandwidth and disk space that is wasted in delivering and storing spam respectively.

Beyond expensive *legal* approaches that seek to unmask the spammers with the goal of prosecuting them [7], the proposed and pursued approaches in the fight against spam can be classified in two broad categories: 1) *sender verification* approaches such as domain keys identified mail (DKIM) [4], and sender ID [9, 11] that aim to answer the question: "Did the sender actually send this email?" and can help in pinpointing the spammers and 2) *mitigation* approaches [2, 13, 12, 8, 14, 6, 5] that either attempt to deter spammers, or filter incoming or outgoing emails for spam. The sender verification approaches rely on being able to make infrastructural changes to the domain name system (DNS) which limits their immediate deployability. Also, their success hinges on the security of the DNS itself, for which solutions are still being worked on. In this work, we focus on the mitigation techniques due to their deployability. In particular, *the goal of the work presented in this paper is to develop an effective spam mitigation approach for organizations that is immediately deployable and does not require any cooperation from anyone outside the organization*. Our approach is complementary in nature to several of the existing mitigation techniques. In particular, the mitigation techniques that throttle spam at the senders [14], use greylisting [8][1], or impose quotas on senders through peer cooperation [2, 13] can be used in addition to our approach. On the other hand, other mitigation techniques that employ collaborative spam filtering based on social email networks [6], predict spam patterns [5], or prioritize email filtering for good mail over spam [12] are competing approaches to ours.

Our approach consists of two techniques. The first technique, *token-based authentication*, allows implementing sender's mail server to put a receiver specific *authentication token string* (referred to as *token* subsequently) in the header of every outgoing email. Replies to such emails would contain the token even if the recipient does not implement the technique if a standard field that gets copied in the reply is used to put the token[2]. Presence of such tokens in incoming mails is then used to identify valid emails which prevents much of the good mail from being subjected to the filtering process. This technique enables email sender's authentication only with simple token management and exchange unlike existing email authentication schemes such as

S/MIME [10] and pretty good privacy (PGP) [1] which require expensive operations and face deployability issues. The second technique we propose, *history-based prioritization*, complements the first technique and is designed for the rest of the incoming email that does not contain tokens. It builds on the email prioritization scheme proposed in [12] and utilizes past information about the sending mail servers, along with the information about their domains to prioritize email filtration. It reduces delays in the delivery of good email over spam because sophisticated spam filtration, which uses a combination of machine learning techniques such as Bayesian filtering [3], DNS-based blacklists of known offenders, and whitelists of good senders, can introduce substantial delays in mail delivery. The design of both the proposed techniques allows any mail server to deploy them independent of the choice made by the any other mail server in the Internet, making them immediately deployable. We evaluate the efficacy of the proposed techniques by analyzing 7 months of departmental email logs.

The rest of this paper is organized as follows. Section 2 describes the data used in analyzing the effectiveness of our approaches. Both our techniques, along with the analysis of each using logs, are presented in Section 3. Finally, Section 4 offers concluding remarks.

2 Data Used in Analyzing Our Approaches

In order to analyze the effectiveness of our approaches, we collected 7 months of Sendmail logs from Indiana University's Computer Science Department, starting April 10th, 2005. An entry in our log is created for each unique email message received and contains the timestamp, IP address and name (if the reverse DNS lookup by the department's mail server was successful) of the sending MTA, anonymized information about the senders and receivers of that email, and a status code for the mail. The status code indicates whether the mail was perceived as a spam or not according to the filtering program. In addition to the data on incoming SMTP connections, we also have anonymized information about all the outgoing mails in the department. Table 1 shows an overview of the data available to us. We exclude all mails where both the sender and receiver are local to focus only on emails that could potentially be spam.

3 Our Approach

Email filtration is a double edged sword: the more sophisticated a software becomes, the lower the false positives and negatives, but the higher the processing times. Table 2 shows the processing times for the 130,122 emails received at our department's mail server for a week. These times range from 0.32 seconds to 3559.56 seconds. Due to the large range the mean, median, and standard deviation are 3.81, 1.48, and 21.63 seconds respectively. It is noteworthy that even though 76.8% of the emails took less than or equal to 3 seconds to finish the filtering process, 5.7% took greater than 10 seconds to process!

Table 1: Overview of the data.

Duration of logs	211 days
Number of *incoming* SMTP connections	3,415,219
Number of *outgoing* SMTP connections	806,215
Number of unique sending MTAs	879,670
Sending MTAs that are *DNS resolved*	505,443
Number of unique sending domains	25,760

Table 2: Processing time distribution.

Range (sec)	Number of messages	% of messages
0.0-1.0	29,094	22.4
1.0-2.0	55,892	43.0
2.0-3.0	14,817	11.4
3.0-4.0	9,448	7.3
4.0-5.0	4,406	3.4
5.0-6.0	2,825	2.2
6.0-7.0	2,041	1.6
7.0-8.0	1,646	1.3
8.0-9.0	1,374	1.1
9.0-10.0	1,104	0.8
> 10.0	7,475	5.7

Our approach aims to prioritize filtration of good emails over spam by using a divide and conquer strategy. We first separate bulk of the good mail using a token-based authentication scheme which we describe in Section 3.1. For the rest of the emails, we use history information about the sending MTA and its domain to predict the goodness/badness of the incoming mail.

3.1 Token-based Authentication

Token-based authentication is an authentication scheme to identify emails from valid senders that a user expects to hear from. It enables a mail server to deliver such mails to its users without subjecting them to the spam filtering process. An MTA deploying this scheme assigns a unique token for each (sender, receiver) pair. The token

is nothing but an alphanumeric string of a small number of bytes, say 64. For each outgoing email, the sending MTA, S, picks the relevant tokens for that sender. For example, if the mail was sent to just one receiver, the sending MTA puts the corresponding token for that receiver in the email header and if the mail is destined for multiple receivers, tokens for each individual receivers are put. Mailing lists can be allocated unique tokens to avoid increasing the email size substantially due to tokens.

This token is returned back to the sender if the receiver R chooses to reply to sender's email. Upon receiving such replies, S can infer that those are not spam by simple comparing the token in the mail to the one it has stored locally, thus delivering such mails immediately without subjecting them to the filtering process. Notice that the tokens accomplish more than what the sending MTA can accomplish simply by keeping track of the receivers for all outgoing mails. This is because sender information in incoming mails can be forged.

The token-based authentication scheme can be deployed even when receiver R does not implement the scheme. Today, this can be accomplished if an MTA uses the *Message-ID* field of the outgoing email header to insert the tokens. This field is copied by most replying MTAs into the *In-Reply-To* field of the reply messages. Thus, emails users expect to get can be separated from the others without any change to the receivers.

The above discussion assumes that the sending MTAs choose tokens for all the (sender, receiver) pairs and the same tokens are retained for all subsequent communications. This scheme can be misused by the spammers who have access to email logs over the Internet. Thus, the assigned tokens should be periodically changed to avoid the security vulnerability of a long lived token. Further, in order to keep the number of tokens bounded, some mechanism for expunging unused tokens would have to be devised.

Alternate schemes for token assignment are possible. For example, instead of having a token per (sender, receiver) pair, the token could be per sender, or per receiver. Both of these schemes reduce the number of tokens an implementing MTA has to keep track of but suffer from individual shortcomings. The sender-based token scheme prevents the senders from being able to specify which receivers they regularly communicate with, something our proposed scheme can easily be extended to do. Similarly, the receiver-based token scheme implies that there will be one token per receiver, irrespective of how many senders communicate with that receiver. This would also prevent senders from specifying which receivers they would like to maintain tokens with because even the same receiver could have diverse meanings to different senders.

3.1.1 Effect of Token-based Authentication

To see how many incoming emails would the token-based authentication scheme impact, we analyzed the data on incoming and outgoing SMTP connections (described in Section 2). Essentially, using the sender and receiver information contained in each incoming and outgoing connection, we extracted the information on unique (sender, receiver) pairs, and the pairs where both parties communicated with each other. Incoming mails with the latter would potentially contain tokens. Table 3 shows the number of pairs observed in our data and the number of SMTP connections that would be affected by the token-based authentication scheme.

Table 3: Incoming mails potentially benefited by token-based authentication.

Number of unique (sender, receiver) pairs	3,051,559
Unique pairs with 2-way communication	12,118
Percentage	0.4%
Total incoming SMTP connections	3,415,219
Incoming connections with tokens	105,891
Percentage	3.1%

Although it appears that the results in Table 3 are disheartening, such is not the case. This is because the small percentage of mails containing the tokens are likely to be the most important and urgent. And these are exactly the ones whose delays are avoided through the use of the token-based authentication scheme. Moreover, the total number of unique pairs are likely to be unusually high due to the use of distinct sender names in spam ($2,988,519$ of the total $3,051,559$ sender/receiver pairs were the kind where an incoming message was not responded to by the recipient), many of which are regularly forged. Further, our results are likely to be pessimistic due to common practice among the students of our department of forwarding emails to their other accounts, commonly the university-wide account that is accessible from anywhere through a Web-based interface.

Table 3 assumes that if a valid token is contained in an incoming mail, it is a good mail. In the case of incoming emails that contain multiple receivers, the above assumes that the mail would be regarded as good by all receivers. In reality, such emails may have to be subjected to the filtering process. To account for such cases, we also considered *total mails*, where an incoming SMTP connection containing n recipients is considered as n mails. Using this notion of total mails, we found that out of the total $4,468,806$ mails, $108,485$ mails (2.4%), would contain tokens and could be authenticated. Table 4 shows

the remaining data on incoming emails that we consider for the history-based analysis.

Table 4: Data after removing emails authenticated by tokens.

Duration of logs	211 days
Number of *incoming* SMTP connections	3,309,328
Number of unique sending MTAs	876,346
Sending MTAs that are *DNS resolved*	502,493
Number of unique sending domains	25,259

3.2 History-Based Prioritization

Even though the token-based authentication scheme is able to prioritize some good mails over others, it accounts for only 3.1% of the incoming SMTP connections. For the rest of the mails, we explore history-based prioritization whose goal is to use past behavior of mail servers and their domains to decide which mails should be prioritized by the spam filters. This allows the likely good mails to be delivered to the recipients faster than a first come first served (FCFS) filtering policy would allow.

The basic notion of history-based prioritization has been examined earlier. In [12], authors utilize the past history of the sending MTAs (referred to as *server history* subsequently) to prioritize which mails should be seen by the spam filters first. They consider an MTA worthy of prioritizing mails from if it sent at least 50% good emails in the past. Using this simple strategy on 69 days of email logs, they achieved accuracies of $74-80\%$ for good mails, $93-95\%$ for spam and viruses, and aggregate accuracies of at most 90%. Also, they found that maintaining a history of $100,000$ past SMTP connections was sufficient to achieve these accuracies.

We explore history-based prioritization in a more general sense with an aim to improve the prioritization accuracies for good mails. Also, while work in [12] focused only on mail servers that sent 10 or more emails, we wish to achieve good results for all MTAs, including the ones that sent fewer than 10 emails. In particular, we first seek the answers to the following questions to find out the parameters that can be used by a history-based prioritization scheme:

- Is the mail more likely to be a spam if the reverse DNS lookup fails on an IP address? We ask this question because many filtering programs compare the domain name contained in an email message to that obtained by the reverse DNS lookup and report it in the email header.
- Is the sending history of a domain indicative of whether the mail is likely to be a spam or not? How does its accuracy compare with the approach used in [12], where only MTA's sending history was utilized?
- Is the number of total mails or average mails per day sent by an MTA or a domain an indicator of whether they mostly send spam or not?
- Is the number of days an MTA is active for a predictor of what kind of mails it would send?
- Does the number of servers per domain predict what kind of emails would an MTA send? We are interested in this question because botnets are often comprised of compromised home machines, who belong to the domains of the service providers, which are relatively fewer in number compared to the total number of domains.

Based on the answers to the above questions, our goal is to incorporate the useful parameters into an algorithm that can then be used by an MTA to prioritize incoming emails for spam filtering. We now find out answers to the above questions one by one.

3.2.1 Reverse DNS Lookup

To explore if any relationship exists between the success of reverse DNS lookup on the name of the sending MTA and the probability of the mail being spam, we compared the good and junk mails sent by MTAs for whom the name resolution fails and the ones for whom the reverse DNS resolution succeeds. The data used in conducting this analysis is from Table 4 and the results are shown in Table 5.

Table 5: Good and junk mail statistics for MTAs.

		unique MTAs	%	SMTP connections	%
unresolved	good only	24,484	2.79	45,090	1.36
	junk only	340,287	38.83	829,779	25.07
	both	9,082	1.04	121,600	3.67
	subtotal	*373,853*	*42.66*	*996,469*	*30.11*
resolved	good only	42,616	4.86	255,774	7.73
	junk only	447,137	51.02	1,069,925	32.33
	both	12,740	1.45	987,160	29.83
	subtotal	*502,493*	*57.34*	*2,312,859*	*69.89*
	total	876,346	100	3,309,328	100

The results presented in Table 5 indicate that over 91% of MTAs for whom reverse DNS lookup fails (38.83% of all MTAs) actually established SMTP connections to

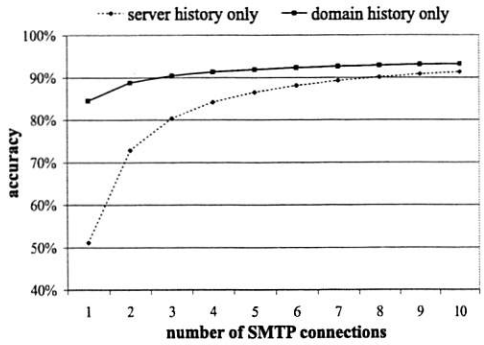

Figure 1: A comparison of using server history versus domain history for MTAs that sent 10 or more emails.

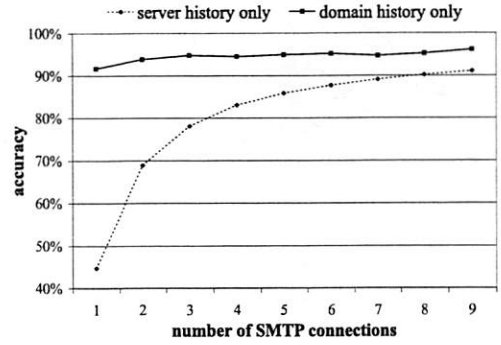

Figure 2: A comparison of using server history versus domain history for MTAs that sent less than 10 emails.

send only junk emails, where a junk email is either a spam or a virus. The corresponding SMTP connections are 83% of the total SMTP connections established by such MTAs. Alternately, only 6.5% of the MTAs for whom name resolution fails established SMTP connections to send only good emails, and only 2.43% of such MTAs actually sent emails that the spam filter perceived as both good and junk. *This implies that the failure of reverse DNS lookup is a strong indicator of what kind of email would an MTA send.*

3.2.2 Domain History

We now use the data in Table 4 to investigate if the history of sending MTA's domain, either by itself or in conjunction with server history, can be used to predict the nature of an incoming SMTP connection. Unlike [12], where the first email from an MTA was considered to be bad (leading to a higher accuracy for junk mails at the cost of good mails which are smaller in percentage), we consider the first incoming mail from an MTA to be bad only if the reverse DNS lookup fails. Otherwise, the incoming mail is considered good. We now observe the overall prediction accuracies when just the domain history and just the server history are used for prediction. The algorithm we use in computing history is: consider an incoming email to be bad if 50% of the past mails sent by the MTA or domain were bad, and good otherwise.

Figures 1 and 2 show the average prediction accuracy for incoming emails as more domain and server history are available (after seeing about 11 emails from a particular domain or MTA, the accuracy seems to stabilize). We split the data into servers that sent greater than or equal to 10 messages overall in our log and those that sent less than 10 mails to see the difference in accuracy results for servers that sent a small number of messages versus those who sent larger number of messages. This is because 91.6% of messages sent by MTAs in the latter set are spam (compared to 50% in the former case) and overshadow the aggregate accuracy results.

Figures 1 and 2 seem to indicate the domain history is a better predictor of the nature of an incoming mail. However, one needs to be careful before reaching this conclusion because the average prediction accuracy numbers are heavily biased by junk messages, which far outnumber the good emails. At this point, the only conclusion we can draw is that perhaps a combination of both server and domain history would be a better predictor of the nature of an email than just the server history, which was used in [12].

The first email from an MTA about which nothing is known deserves special attention. This is because such emails comprise 26.5% of the total incoming emails according to Table 4. While work in [12] assumes all such *first time emails* to be bad, we have so far used the results from our reverse DNS lookup failure analysis and assumed that only the first time emails from MTAs for whom the reverse DNS lookup fails are bad. We now explore if the use of domain history of the sending MTA, where available, helps predict the first time emails from unknown MTAs better. Table 6 compares the accuracy of first time emails for 1) assumption in [12] (all first time emails are junk), 2) our previous assumption (a first time email is bad if reverse DNS lookup fails), and 3) our first assumption along with the use of domain history where available. *The results show that using domain history improves the accuracy of good first time mails without compromising the overall prediction accuracy or the prediction accuracy of junk mails.*

3.2.3 Average Mails per Day and Total Mails

We now explore the relationship between the number of total and average mails sent by an MTA and the probability that the next mail from that MTA is spam. For each MTA, the average mails per day is computed by dividing the total mails it sent by the number of days it had been *active* for. We define active duration for an MTA to be

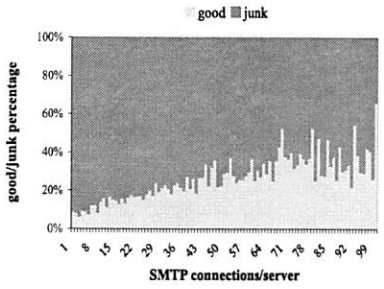

Figure 3: Relationship between total mails sent by an MTA to the percentage of good/junk mails it sent.

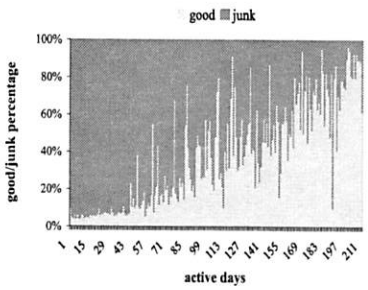

Figure 4: Relationship between active days of an MTA to the percentage of good/junk mails it sent.

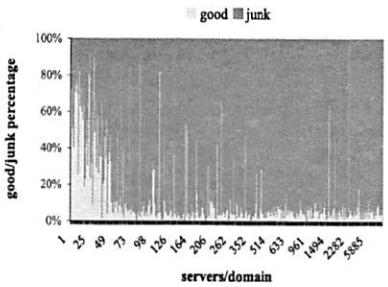

Figure 5: Relationship between number of MTAs per domain to the percentage of good/junk mails each sent.

Table 6: Prediction accuracy for first time emails from MTAs.

	All bad	Bad if DNS resolution failed	Use domain history where available
good mails	0%	62.54%	23.51%
junk mails	100%	43.18%	97.23%
average	90.89%	44.94%	92.00%

the number of days from when we witness a mail from it the first time to the last time it establishes an SMTP connection. *Figure 3 indicates that MTAs with relatively fewer total emails sent more spam than good mails.* The results for average mails per day were inconclusive and are omitted due to space constraints. Further, notice that using average and total mails in email prioritization is tricky because these numbers can change over time. We utilize the available past information to compute these numbers in the algorithm we design subsequently.

3.2.4 Active Days

A related parameter to the total number of mails is the duration for which a server stays active. Figure 4 shows the relationship between the number of days an MTA stays active in our log and the percentage of good/junk mail it sends. *We conclude that the longer an MTA stays active, the less spam it sends.*

3.2.5 Servers per Domain

Figure 5 shows the relationship between the number of MTAs in a domain and the percentage of good and junk mails. *It shows that the more the number of MTAs belonging to a domain, the more the likelihood that those MTAs will send more junk mails than good ones.* This is intuitive since much of the spam today is sent by compromised machines that are part of large botnets comprising of hundreds of thousands of machines. And many of the compromised machines belong to home users, who subscribe to handful of the popular big Internet service providers (ISPs).

3.2.6 Using Above Parameters in an Algorithm

With the goal of predicting incoming good emails as accurately as possible, we now use server history information and the above parameters to design a prediction algorithm for prioritizing emails for filtration. The algorithm presented in algorithm 1, computes the probability P_i that an incoming mail i is good in three separate cases: 1) when no server history information is available, 2) when server history is between 0.4 and 0.6 (implying the server sends both good and junk mails with close enough probabilities – this special case is required for servers like *yahoo.com* that aggregate mails from all kinds of users), 3) when server is either less than 0.4 (implying it sends junk mails most of the time) or greater than 0.6 (implying that it sends good mails most of the time). An incoming mail is considered to be good if $P_i \geq 0.5$. After the prediction of each incoming mail, the server and domain histories (referred to as *good mail probabilities*) for the sending MTA and its domain are updated according the following:

$$GMP(M_i) = \frac{N_{good}(M_i)}{N_{total}(M_i)} \quad (1)$$

$$GMP(D_i) = \frac{N_{good}(D_i)}{N_{total}(D_i)} \quad (2)$$

In equations 1 and 2, N_{good} and N_{total} are the numbers of good and total SMTP connections seen so far from MTA M_i and domain D_i respectively.

Parameters ρ, ϵ, and τ used in our algorithm are for total mails from mail i's server, relative days server for mail i was active for with respect to log duration at the time of measurement, and number of servers seen from the same domain as mail i's server respectively. They are chosen to be 10, 0.6, and 50 respectively based on our

Table 7: Performance comparison of our algorithm with the server history algorithm presented in [12].

	MTAs that sent 10 or more mails		MTAs that sent less than 10 mails		All MTAs	
	server history only	our algorithm	server history only	our algorithm	server history only	our algorithm
good	85.94%	95.39%	33.49%	35.37%	80.40%	90.10%
junk	87.43%	77.09%	99.04%	98.29%	93.49%	89.14%
average	86.42%	86.71%	92.34%	92.85%	89.39%	89.79%

Algorithm 1 Our algorithm.

if no history information available for M_i **then** {// case 1}
  if no domain history information available for D_i **then**
    if M_i's reverse DNS lookup failed **then**
      $P_i = 0.0$ {// mail is junk}
    else
      $P_i = 1.0$ {// mail is good}
    end if
  else
    $P_i = \gamma * GMP(D_i)$ {// γ allows tuning}
  end if
else if $0.4 \leq GMP(M_i) \leq 0.6$ **then** {// case 2}
  if the previous mail from M_i was good **then**
    $P_i = 1.0$
  else {// consider weighted average of server and domain histories}
    $P_i = \alpha * GMP(M_i) + \beta * GMP(D_i)$
    if $AD_i > \epsilon$ **then** {// consider days server is active for}
      $P_i = \lambda * P_i$ {// λ allows tuning}
    else if $SD_i > \tau$ **then** {// consider number of servers per domain}
      $P_i = \delta * P_i$ {// δ allows tuning}
    end if
  end if
else {// case 3}
  if $TM_i < \rho$ **then** {// consider weighted average if total mails from this server $< \rho$}
    $P_i = \alpha * GMP(M_i) + \beta * GMP(D_i)$
  else {// otherwise only consider server history}
    $P_i = GMP(M_i)$
  end if
end if

log. The tunable parameters γ, α, β, λ, and δ are chosen to be 0.7, 0.3, 0.7, 1.3, and 0.8 respectively. These are chosen to maximize the accuracy of prediction of good mails over junk mails, while ensuring that the overall prediction accuracy across all types of mails is not compromised. The latter is important to ensure that the spam filters do not end up dealing with much junk mail while processing the good mails.

Table 7 and Figure 6 show the accuracies of prediction resulting from our algorithm and compares it to the server history information algorithm used in [12]. Though we set the above mentioned parameters using the log, we do not explicitly use a training phase for either of the algorithms and instead evaluate their effectiveness as sending information becomes available. We show three view points in Table 7, 1) aggregate accuracies across all types of MTAs, 2) accuracies for MTAs that sent 10 or more emails, and 3) accuracies for MTAs that sent less than 10 emails. Figure 6 shows only the second view of point. Overall, our algorithm substantially outperforms the server history algorithm in prioritizing good emails for all MTAs, and for MTAs that sent 10 or more emails. In all other categories, we perform at least as well. In comparison, Return Path[3], a company that monitors email performance for online marketers, estimates that current spam filters misclassify nearly 19 percent of good email as spam.

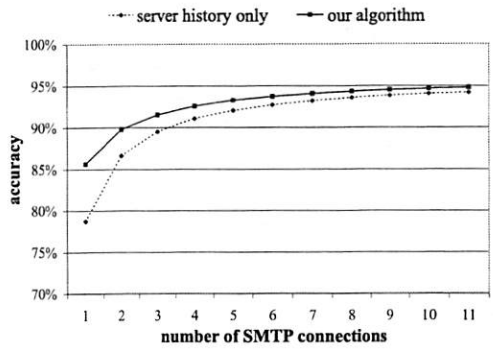

Figure 6: Comparison of prediction accuracy of our algorithm with algorithm presented in [12] as number of SMTP connections increases (for MTAs with greater than or equal to 10 SMTP connections).

4 Concluding Remarks

We have shown that the combination of token-based authentication and history-based prediction can help in delivering good mails to their recipients much faster than spam filtering alone. We believe that our history-based prediction algorithm does as best as an algorithm can do. The reason for this is that some MTAs (39% in our data) consistently sent both types of mails. These appear to be MTAs like *hotmail.com* that serve users with varying intentions, perhaps including spammers.

References

[1] ATKINS, D., AND ET AL. Pretty Good Privacy. RFC 1991, Aug. 1996.

[2] BALAKRISHNAN, H., AND KARGER, D. Spam-I-am: A Proposal for Spam Control using Distributed Quota Management. In *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)* (Nov. 2004).

[3] A plan for spam. `http://www.paulgraham.com/spam.html`.

[4] Domain Keys Identified Mail (DKIM). `http://mipassoc.org/dkim/`.

[5] GOMES, L. H., CASTRO, F., ALMEIDA, V., ALMEIDA, J. M., ALMEIDA, R. B., AND BETTENCOURT, L. Improving spam detection based on structural similarity. In *USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet* (July 2005).

[6] KONG, J., BOYKING, P., RCZACI, B., SARKAR, N., AND ROYCHOWDHURY, V. Scalable and reliable collaborative spam filters: harnessing the global social email networks. In *Conference on Email and Anti-Spam* (July 2005).

[7] KORNBLUM, A. Searching for John Doe: finding spammers and phishers. In *Conference on Email and Anti-Spam* (July 2005).

[8] LEVINE, J. Experiences with greylisting. In *Conference on Email and Anti-Spam* (July 2005).

[9] Microsoft's SenderID framework. `http://www.microsoft.com/mscorp/safety/technologies/senderid/default.mspx/`.

[10] S/MIME working group. `http://www.imc.org/ietf-smime/`.

[11] Sender Policy Framework. `http://spf.pobox.com/`.

[12] TWINING, R. D., WILLIAMSON, M. W., MOWBRAY, M., AND RAHMOUNI, M. Email prioritization: reducing delays on legitimate mail caused by junk mail. In *USENIX Annual Technical Conference (USENIX)* (July 2004).

[13] WALFISH, M., ZAMFIRESCU, J., BALAKRISHNAN, H., KARGER, D., AND SHENKER, S. Distributed Quota Enforcement for Spam Control. In *3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (May 2006).

[14] ZHONG, Z., HUANG, K., AND LI, K. Throttling outgoing spam for Webmail services. In *Conference on Email and Anti-Spam* (July 2005).

Notes

[1] Mail servers using greylisting temporarily reject any email from senders that they do not recognize under the assumption that legitimate mail servers will retry later.

[2] An example of such a field is the message identifier put by the mail server in the *Message-ID* field of the email header. This field is copied into the *In-Reply-To* field in the replies to this message by most mail clients.

[3] http://www.returnpath.net.

Strider Typo-Patrol: Discovery and Analysis of Systematic Typo-Squatting

Yi-Min Wang, Doug Beck, Jeffrey Wang*, Chad Verbowski, and Brad Daniels
Microsoft Research, Redmond * *PCRethinking.com*

Abstract

Typo-squatting refers to the practice of registering domain names that are typo variations of popular websites. We propose a new approach, called Strider Typo-Patrol, to discover large-scale, systematic typo-squatters. We show that a large number of typo-squatting domains are active and a large percentage of them are parked with a handful of major domain parking services, which serve syndicated advertisements on these domains. We also describe the Strider URL Tracer, a tool that we have released to allow website owners to systematically monitor typo-squatting domains of their sites.

1. Introduction

Typo-squatting refers to the practice of registering domain names that are typos of their ***target domains***, which usually host websites with significant traffic. The individuals or organizations who register ***typo-squatting domains*** (or ***typo domains***) are referred to as ***typo-squatters***. Some major typo-squatters are known to have registered thousands or more of typo domains [1,2,3].

Web traffic generated through typo-squatting is unwanted for many reasons. From the users' perspective, such ***typo traffic*** often startles them with unexpected results, followed by an annoying barrage of pop-up and pop-under advertisements (ads). There is a documented incident where a typo domain of a popular website was serving vulnerability-exploiting scripts to install malware [4,5]. Some typo domains of children's websites have been observed to redirect to or link to adult websites, endangering Internet safety by potentially exposing minors to harmful material [6,7].

From the business perspective, many of the typo-squatting cases involve "bad-faith" domain registrations or trademark violations [8,9,10]. Worse yet, it is not uncommon to see a typo domain displaying ads from competitors of the target-domain owner or even negative ads against the owner (e.g., investment-loss law firm's ads on typos of brokerage firms). In other cases, some advertisers are unwillingly paying for their ads being served on typo domains of their own websites, because such traffic is intended to go directly to their sites in the first place [11].

In this paper, we describe the Strider Typo-Patrol System for discovering and analyzing typo domains. Our patrol results reveal that a large percentage of typo domains are ***"parked"*** with a handful of major ***domain parking services***. Domain parking is a special case of advertisement syndication: while the latter attempts to serve relevant contextual ads based on the publishers' web content, the former serves ads based on merely the domain name because parked domains typically have no content. We show that many typo-squatters are taking advantage of the domain-parking infrastructures to perform large-scale, systematic typo-squatting. However, by doing so, they also expose their typo domains to systematic discovery enabled by monitoring and analyzing ads-fetching traffic sent to the parking services.

The paper is organized as follows. Section 2 describes how domain parking works and discusses statistics related to the amount of unwanted traffic potentially generated through typo-squatting. Section 3 presents the Strider Typo-Patrol System. Section 4 analyzes typo-patrol data and quantifies the prevalence of typo-squatting through domain parking. Section 5 describes the Strider URL Tracer designed to provide visibility and control over typo traffic. Section 6 surveys related work, Section 7 discusses remaining issues, and Section 8 concludes the paper.

2. Understanding Domain Parking

Advertisement syndication refers to the business practice of serving ads by instructing the client-side browser software to fetch ads from an ads server and compose them with the content of the website that the user intends to visit. Syndication is typically implemented using the browser's third-party URL mechanism: when a user visits a ***primary URL*** (hosted by the first party) either by typing the URL into the browser address bar or by clicking a link on a web page, the browser may be instructed by the content returned by the primary-URL page to automatically visit one or more ***secondary URLs*** hosted on third-party servers, without explicit knowledge or permission from the user. We refer to these secondary URLs as ***third-party URLs*** in this paper. These third-party URLs usually contain information about the primary URL so that the syndicators can serve the most relevant ***contextual ads*** based on the primary-URL page content and potentially the historical information about the visiting machine or user.

Domain parking is a special case of advertisement syndication: the primary URL is a ***parked domain*** that does not contain any real content and syndicated domain-parking ads, usually in the form of ads listings, become the main content of the page displayed to the user. In order to attract sufficient traffic for serving ads,

parked domains are usually domains with well-known generic names [12] or typo domains of popular websites. See [13] for screenshots of sample domains parked with various parking services.

Next, we use two actual examples to illustrate how typo-squatting through domain parking is typically implemented using third-party URLs. When a browser visits http://disneychannell.com, it receives a response page containing a frame that loads http://www.sedoparking.com/disneychannell.com. This URL is responsible for serving the main domain-parking ads listing. The basic idea of Strider Typo-Patrol is to *scan a large number of typo domains, monitor all third-party URL traffic, and group the domains by the behind-the-scenes domain parking servers in order to facilitate investigation and prioritize actions.*

Some domain parking services provide additional information in their third-party URLs that enables further analysis. For example, when a browser visits http://disneyg.com, the response page contains a frame that loads

http://apps5.oingo.com/apps/domainpark/domainpark.cgi?s=disneyg.com&dp_lp=24&hl=en&dp_lp=7&cid=DTRG4295&dp_p4pid=oingo_inclusion_xml_06&dp_format=1.3

where the "cid" field appears to contain a Client ID that uniquely identifies a typo-squatter. In Section 4, we show how this information enables us to quickly discover thousands of typo domains that are registered to a well-known, serial typo-squatter [2,14].

Domain parking services provide convenient and effective contextual-ads infrastructures that make even marginal typo domains profitable [15]. With the annual domain registration fee as low as $7.00 [16], a rule-of-thumb figure for pay-per-click programs is that a parked typo domain only needs to attract between *one unique visitor every two days* and *two visitors per day* (depending on the pay-out levels) to generate sufficient income to cover the fee. (As a reference, http://slsahdot.org records statistics of tens of hits per day.) According to alexa.com on March 12, 2006, the servers owned by the top two domain parking services identified in our study were reaching between 3,300 and 5,200 per million users daily and their servers had a traffic rank between #221 and #438. These numbers are comparable to those for popular websites such as travelocity.com (#248), orbitz.com (#315), usatoday.com (#347), and slashdot.org (#375). Although many parked domains may be generic-name domains, the fact that we were able to discover thousands of parked typo domains within a short time through simple automated searching does provide evidence that unwanted traffic due to parked typo domains could be significant.

3. Strider Typo-Patrol System

The Strider Typo-Patrol System provides automatic scanning and systematic analysis of typo domains. It consists of three main components: a *typo-neighborhood generator*, a *typo-neighborhood scanner*, and a *domain-parking analyzer.*

3.1. Typo-Neighborhood Generation

Given a target domain, we define its ***typo-neighborhood*** as the set of URLs generated from the following five typo-generation models, which are commonly used in the wild:

(1) Missing-dot typos: The "." following "www" is removed, e.g., wwwSouthwest.com.

(2) Character-omission typos: Characters are omitted one at a time, e.g., Diney.com and MarthStewart.com.

(3) Character-permutation typos: Consecutive characters are swapped one pair at a time, unless they are the same characters, e.g., NYTiems.com.

(4) Character-replacement typos: characters are replaced one at a time and the replacement is selected from the set of characters adjacent to the given character on the standard keyboard, e.g., DidneyWorld.com and USATodsy.com.

(5) Character-insertion typos: characters are inserted one at a time and the inserted character is chosen from the set of characters adjacent to either of the given pair on the standard keyboard (and including the given pair), e.g., WashingtonPoost.com and Googlle.com.

3.2. Typo-Neighborhood Scanning

The Typo-Patrol scanner is an extension of our previous Strider HoneyMonkey scanner [5]. Given a typo-neighborhood list, it launches a browser to visit each domain and records *all secondary URLs visited and their ordering, the content of all HTTP requests and responses, and optionally a screenshot.*

3.3. Domain-Parking Analysis

We currently perform three types of analysis on the typo-neighborhood scan data:

(1) Given a target category and the lists of typos of target domains in the category, we analyze how heavily the category is being typo-squatted and which domain parking services are the major players. Specifically, we group the scanned typo domains by the parking services they generated third-party traffic to, and highlight those services that are behind a large number of typo domains.

(2) Given the typo-patrol results of a trademarked target domain, we perform a similar analysis to identify those major parking services with which the trademark owner may want to file complaints. In some cases, it is more effective to go after the typo-squatters who actually purchased the typo domains than to complain to parking services which are only responsible for profiting from serving ads on those domains. We use two additional pieces of information to further divide and rank typo domains parked with a single service in order to help trademark owners prioritize their actions against typo-squatters.

The first piece of information is the Client ID field mentioned in Section 2. The second piece of information is the ***anchor domain*** that is used to aggregate traffic from multiple typo domains to simplify operations and to enable scalable typo-squatting. For example, tens of typo domains of NationalGeographic.com were "funneling" traffic through the same anchor playbov.com; typo domains LaSalleBanl.com and SovererignBank.com are sharing the same anchor baankaccount.com. We found that, in most cases, typo domains sharing the same anchor are registered to the same WhoIs registrant [17]. By grouping those typo domains that first redirect to the same anchor domain before generating traffic to the parking service, we eliminate the need to investigate each individual domains.

(3) For analyses that require searching for specific keywords (e.g., sexually-explicit keywords used in the analysis in Section 4.4), we analyze the HTTP response pages and extract all typo domains with a match.

4. Typo-Patrol Data Analysis

We first present two kinds of analysis to assess the prevalence of typo-squatting and to identify major domain parking services that are involved: ***vertical analysis*** uses a single type of typos for a large number of target domains; ***horizontal analysis*** uses multiple types of typos for a smaller set of target domains. Then, we present a case study in which we identified thousands of typo domains owned by a well-known typo-squatter. Finally, we investigate typo domains of children's websites that serve questionable ads.

4.1. Missing-dot Typos of Top 10,000 Sites

For the vertical analysis, we scanned the missing-dot typos of the 10,000 most popular domains. Our result showed that 5,094 (***51%***) of the 10,000 typo domains were active at the time of the scan. Figure 1 ranks the top six domain parking services by the number of typo domains that serve ads from them. We make the following observations: (1) the top two parking services clearly stand out, each covering approximately ***20%*** of active typo domains; (in addition, note that sedoparking.com uses the same ads-serving infrastructure as oingo.com according to http://www.sedoparking.com); (2) the top six parking services together account for more than half (***59%***) of the active domains and ***30%*** of all the artificially generated missing-dot typo domains.

4.2. Typo-Neighborhoods of Popular Sites and High-Risk Phishing Targets

For the horizontal analysis, we selected two sets of target domains: the first set consists of 30 of the most popular sites according to alexa.com; the second set consists of 30 high-risk targets by phishing attacks, selected from [18]. For each target domain, we scanned its typo-neighborhood composed of typo domains generated from all five typo-generation models. The two sets of results are shown in Figure 2 and Figure 3, respectively.

	Parking service	# typos parked	% of active (5,094)	% of all (10,000)
#1	Information.com/ Domainsponsor.com	1,082	21%	11%
#2	Oingo.com	992	20%	9.9%
#3	Sedoparking.com	439	8.6%	4.4%
#4	Qsrch.com	227	4.5%	2.3%
#5	Netster.com	146	2.9%	1.5%
#6	Hitfarm.com	109	2.1%	1.1%
	Total	**2,995**	**59%**	**30 %**

Figure 1. Top six domain parking services in the missing-dot typo-neighborhoods of top 10,000 websites

	Parking service	# typos parked	% of active (2,233)	% of all (3,136)
#1	Oingo.com	420	19%	13%
#2	Information.com/ Domainsponsor.com	306	14%	9.8%
#3	Sedoparking.com	74	3.3%	2.4%
#4	Qsrch.com	74	3.3%	2.4%
#5	Hitfarm.com	69	3.1%	2.2%
#6	Netster.com	50	2.2%	1.6%
	Total	**993**	**44%**	**32%**

Figure 2. Top six domain parking services in the typo-neighborhoods of 30 most popular websites

	Parking service	# typos parked	% of active (1,596)	% of all (3,780)
#1	Oingo.com	695	44%	18%
#2	Information.com/ Domainsponsor.com	292	12%	7.7%
#3	Netster.com	66	4.1%	1.7%
#4	Sedoparking.com	60	3.8%	1.6%
#5	Hitfarm.com	37	2.3%	1.0%
#6	Qsrch.com	28	1.8%	0.7%
	Total	**1,178**	**67%**	**31%**

Figure 3. Top six domain parking services in the typo-neighborhoods of 30 high-risk phishing targets

We make the following observations: (1) in the two sets of scans, ***71%*** (2,233/3,136) and ***42%*** (1,596/3,780) of the generated typo domains were active, respectively; (2) the top six parking services remain the same across all three sets of data except for minor re-ordering of rankings; (3) again, the top two parking services stand out, even more so than in Figure 1; (4) the overall numbers for the top six services remain fairly consistent: they together account for ***40% to 70%*** of active typo domains and around ***30%*** of all generated typos.

4.3. Case Study: A Large-Scale Typo-Squatter

A major typo-squatter has been observed to perform systematic typo-squatting on many target domains [2,14]. But there has been no estimate of how big its typo-squatting business is. Since it has been changing its registrant name in the WhoIs records, we will refer to the company as *DomainSquatter* in this paper. During our investigation, it became clear that DomainSquatter was parking a lot of domains with oingo.com and it was using anchor domains heavily. By analyzing traffic aggregation through tens of anchor domains in the horizontal analysis, we were able to identify the two Client IDs used by DomainSquatter, one for the typo domains and the other for the anchors. We then extracted all scanned domains parked with oingo.com that were using those two Client IDs and used WhoIs lookups to verify that almost all of them were registered to DomainSquatter.

Figure 4 shows that, among the total of 5,094+2,233+1,596=8,923 active typo domains from the three sets of data, ***2,107 (24%)*** were parked with oingo.com and ***1,607 (18%)*** were registered to DomainSquatter. That is, when a user made a typo and reached an active typo domain, one in every ***four*** such domains would serve ads from oingo.com and one in every ***six*** would profit DomainSquatter if the user clicks the ads. It is also significant to note that DomainSquatter accounted for ***76%*** of the 2,107 typo domains parked with oingo.com. It did not appear to be targeting any specific industry, as speculated in [2]: it was squatting 29 of the 30 target domains in both sets used in the horizontal analysis.

	# owned by Domain Squatter	# typos parked with oingo.com	% typos parked with oingo.com	% of active
Figure 1	732	992	74%	14%
Figure 2	310	420	74%	14%
Figure 3	565	695	81%	35%
Total	1,607	2,107	76%	18%

Figure 4. Large-scale, systematic typo-squatting by a major typo-squatter

Since we started reporting discovered typo domains at http://research.microsoft.com/Typo-Patrol in December 2005, DomainSquatter has been de-registering most of the reported domains almost on a daily basis. First, most of the anchor domains were abandoned (see the consistent traffic drops around mid-December across multiple anchors [19]). Then, the registrant names in most of the WhoIs records were changed [14]. In total, we reported ***2,182*** typo domains owned by DomainSquatter (including the 1,607 domains from the three data sets). Around mid-March 2006, we rescanned those 2,182 domains and found that ***1,668 (76%)*** of them were no longer active. Among the remaining 514 active typo domains, 355 are still parked with oingo.com and 159 are parked with others.

4.4. Typo-Neighborhoods of Children's Websites

In our final set of scans, we performed typo-patrol analysis on 50 popular children's sites. The 50 neighborhoods contained 7,094 typo domains, among which 2,685 (38%) domains were active. By parsing the HTTP responses for sexually-explicit keywords and by manually screening the recorded screenshots to locate other suspects, we found a total of ***110*** (***4.1%*** of 2,685) domains that contained questionable content: four domains redirected to adult sites directly, 36 domains contained at least one conspicuous link to an adult site, and the remaining domains displayed at least one conspicuous adult-category link to a page of adult ads listings.

By analyzing the third-party URL traffic, we found that the top two domain parking services together were responsible for serving ads on ***80%*** of those 110 typo domains: 53 (46.5%) domains parked with oingo.com and 37 (33.6%) domains parked with information.com/domainsponsor.com. Among the 53 domains, 46 were registered to DomainSquatter and an analysis of those 46 domains revealed three safety issues with domain parking.

First, typo-squatters can park an anchor domain with a sexually-explicit name such as http://freexxxlinks.us to "trick" domain parking services into serving questionable ads and then redirect typo domains of children's websites to that anchor so that the ads are displayed to children who made a typo. For example, 20 typo domains of the children's website http://flashplayer.com were redirected to http://freexxxlinks.us [6].

Second, sometimes domain parking services were serving adult ads even on anchor domains that do not have a sexually-explicit name, e.g., http://disnryland.com, which was an anchor for typos of http://kimpossible.com [6]. We speculate that the typo-squatter might have explicitly specified sexually-explicit keywords in order to trick the parking service's contextual-ads algorithm into serving questionable ads.

The third issue is inherent to the fact that domain parking is a special case of advertisement syndication: given merely a domain name like gropvygirls.com [6], the algorithm may not have sufficient knowledge to determine that it is a typo domain of the children's website groovygirls.com, and may make a mistake in deriving the keywords and result in inappropriate advertisements being displayed to children.

Soon after the troubling practice was exposed in mid-December 2005 [3], the two anchor domains http://disnryland.com and http://freexxxlinks.us that together were responsible for 26 of the 110 typo

domains were removed. After the practice attracted public attention again due to our tool release in April 2006 [20], another round of ads cleaning was done to remove questionable ads.

5. Strider URL Tracer with Typo-Patrol

Motivated by the prevalence of typo-squatting, we developed a tool, named ***Strider URL Tracer*** [21], to provide users with visibility and control over third-party traffic, which has mostly remained under the cover for the past decade.

The tracer provides four main functionalities. It supports a ***"URL Scan History"*** view that records the timestamp of each primary URL visited and its associated secondary URLs, grouped by domains. It supports an alternative ***"Top Domain"*** view that, for each secondary-URL domain, displays all the visited primary URLs that generated traffic to it. Domains associated with more primary URLs are displayed closer to the top. For every URL displayed in either of the views, the tool provides a right-click menu with two options: the ***"Go"*** option that allows the URL to be revisited (so that the user can figure out which ad came from which URL) and the ***"Block"*** option that allows blocking of all future traffic to and from that domain.

We envision the URL tracer to be used primarily in three scenarios. The first scenario is an on-demand investigation tool. When a browser user encounters any questionable content from an unknown source, she can use the "browser history patrol" feature to rescan recently visited URLs, use "Go" to determine which URL was responsible for serving the content, and use "Block" to prevent the browser from visiting that domain in the future.

The second scenario is a typo-patrol tool used by trademark owners who want to monitor typo domains. It is often too expensive for target-domain owners to investigate and take actions against a large number of individual typo domains. We have incorporated into the tool a feature that takes a target domain name and automatically generates and scans its typo-neighborhood. The trademark owner can then use the "Top Domain" view to identify those parking services that are heavily involved. This domain parking-based analysis provides an efficient and low-cost solution for the owners to file multi-domain complaints with major parking services (e.g., [22]) to request banning of typo domains from their parking programs. Together with IP address-based grouping, such analysis also facilitates grouping of multiple typo domains that are owned by the same registrant and/or hosted by the same ISP. This makes it easier for trademark owners to file multi-domain disputes against typo-domain registrants and to send multi-domain takedown notices to the hosting ISPs.

In the third scenario, the scanning and data analysis portion of Strider Typo-Patrol can be applied to non-typo questionable domains as well, which may be obtained from reverse IP lookups, DNS zone files, services that monitor new domain registrations, etc. For example, we scanned a list of 3,990 domains, all of which contain "microsoft" (without any typo) in their domain names. Our scan determined that 2,938 of them were active and the six domain parking services identified in this paper together parked 949 domains, or ***32%***. Again, the top two stood out: oingo.com parked 509 (***17%***) domains (of which 351 were linked to DomainSquatter's Client IDs); Information.com/Domainsponsor.com parked 321 (***11%***) domains. This preliminary investigation reveals that certain domain parking services may be profiting directly from well-known brand names, in addition to their typos.

6. Related Work

Domain-name typo-squatting has received increasing attention over the past few years [1,4]. However, the community's understanding of the typo-squatting practice has been mostly based on individual cases through manual and ad-hoc investigations. Our Typo-Patrol work proposes the first automatic and systematic approach to discovering and analyzing typo domains and typo-squatters.

The Fiddler HTTP Debugging Proxy [23] intercepts all browser traffic. It provides more powerful traffic monitoring and control capabilities than the Strider URL Tracer. But it does not provide primary-secondary associations, which are essential for typo patrol.

The domain blocking functionality already exists in a few different forms, but it has not been integrated with the browsing history as an online, on-demand feature. For example, Firefox users can use the userContent.css file to block selected domains [24] and Internet Explorer users typically use the Windows hosts files to block unwanted ads [25]. In contrast with these two mechanisms which are usually used to perform wholesale blocking of all ads, our tool allows the users to see which ad came from which domain and gives them the power to use on-demand domain blocking to discourage advertising companies from serving questionable ads without blocking legitimate ads.

Third-party URLs have been used by malicious websites to execute and install malcode on client machines [5] and by advertising and web analytics companies to implement ***web beacons*** (or ***web bugs***) to track users' browsing behaviors [26]. The Strider URL Tracer can be used to expose those behind-the-scenes exploiters that pretend to be advertisement syndicators, but serve vulnerability-exploiting scripts instead of ads [5]. The "Top Domain" view is particularly useful for exposing web beacons.

The homograph attack is another way to create misleading domain names of popular websites by replacing some of the characters with other visually similar ones, possibly from a different language [27]. A

recent study by Holgers et al. [28] showed that currently homograph attacks are rare and not severe; like typo-squatting domains, most "homographed" domains serve advertisements.

7. Discussions

It is important to note that the typo-squatting domains scanned by Strider Typo-Patrol are generated automatically based on a set of typo-generation algorithms. Final determination of whether they are registered in bad faith or in violation of trademark rules is up to the trademark owners, the parking services, and the domain dispute process. It is possible that an algorithmically generated typo domain happens to be another legitimate domain.

Second, we believe that most domain parking services are legitimate advertising companies. Many of them have stated trademark policies and rules [22,29] but, until now, it has not been an easy task for them to distinguish legitimate domains from typo-squatting domains. We encourage parking services to use our tool to identify systematic typo-squatting domains in their parking programs and to identify large-scale typo-squatters among their customers.

8. Summary

We have described Strider Typo-Patrol for automatic discovery and systematic analysis of typo-squatting domains. By scanning three sets of typo domains and analyzing their third-party URL traffic, we have identified two domain parking services that are particularly active in serving ads on at least thousands of typo domains, and found that the top six parking services are responsible for parking around ***30%*** of all algorithmically generated typo domains and ***40%~70%*** of the active ones. We have discovered thousands of typo domains registered to a well-known, large-scale typo-squatter, who according to our study was responsible for as many as ***76%*** of all typo domains parked with a major parking service oingo.com [30], or ***18%*** of all active typo domains from our scans. This typo-squatter was also responsible for a significant percentage of typo domains of children's websites that were serving questionable ads. We have developed the Strider URL Tracer with Typo-Patrol to help provide visibility into the typo-squatting business practice and to allow owners of popular websites to monitor potential violations of their trademarks.

References

[1] Benjamin Edelman, "Large-Scale Registration of Domains with Typographical Errors," Sept. 2003, http://cyber.law.harvard.edu/people/edelman/typo-domains/.

[2] Will Sturgeon, "Serial typo-squatters target security firms," ZDNet, Sep. 19, 2005, http://news.zdnet.com/2100-1009_22-5873001.html.

[3] Strider Typo-Patrol, http://research.microsoft.com/Typo-Patrol.

[4] "Googkle.com installed malware by exploiting browser vulnerabilities," http://www.f-secure.com/v-descs/googkle.shtml.

[5] Yi-Min Wang, et al., "Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities", in *Proc. NDSS*, February 2006.

[6] Screenshots of questionable advertisements, http://research.microsoft.com/Typo-Patrol/screenshots.htm.

[7] "Truth in Domain Names Act of 2003," http://www.cybertelecom.org/dns/truth.htm.

[8] Anticybersquatting Consumer Protection Act (ACPA), http://www.patents.com/acpa.htm, November 29, 1999.

[9] Uniform Domain-Name Dispute-Resolution Policy (UDRP), http://www.icann.org/udrp/udrp.htm.

[10] "Cybersquatter Fined $100,000 Per Domain Name," http://www.gigalaw.com/articles/2000-all/isenberg-2000-11a-all.html, November 2000.

[11] "Typogoogling," http://www.f-secure.com/weblog/archives/archive-122005.html#00000743.

[12] Domain potential, https://partner.dotzup.com/flush.html.

[13] Screenshots of sample parked domains, http://research.microsoft.com/URLTracer/Parked_Domains.htm.

[14] Numerous domain name dispute cases against Unasi, Inc., http://research.microsoft.com/Typo-Patrol/default.htm#Unasi.

[15] Ryan Naraine, "MS Research: Typo-Squatters Are Gaming Google," eWeek.com, December 19, 2005, http://www.eweek.com/article2/0,1895,1903695,00.asp.

[16] Bulk registration pricing, https://www.godaddy.com/gdshop/registrar/bulkprices.asp?se=%2B&ci=176.

[17] WhoIs lookup, http://domaintools.com or http://whois.ws.

[18] Millersmiles Phishing Scams by Targeted Company, http://www.millersmiles.co.uk/scams.php.

[19] Abandoned anchor domains for oingo-parked typo domains, http://research.microsoft.com/Typo-Patrol/Major_Anchors.htm.

[20] "Microsoft 'URL Tracer' Hunts Typosquatters," Slashdot, http://it.slashdot.org/article.pl?sid=06/04/07/1818228&threshold=-1, April 7, 2006.

[21] Strider URL Tracer with Typo-Patrol, http://research.microsoft.com/URLTracer/.

[22] Google AdSense for Domains Trademark Complaint Procedure, http://www.google.com/tm_complaint_afd.html.

[23] Fiddler HTTP Debugging Proxy https://fiddlertool.com/fiddler/.

[24] Blocking advertisement with the Firefox userContent.css file, http://www.mozilla.org/support/firefox/adblock.html.

[25] Blocking Unwanted Parasites with a Hosts File, http://www.mvps.org/winhelp2002/hosts.htm.

[26] Stefanie Olsen, "Ad firms set rules for Web tracking bugs," CNET News.com, November 26, 2002, http://news.com.com/Ad+firms+set+rules+for+Web+tracking+bugs/2100-1023_3-975385.html?tag=st.ref.goo.

[27] Evgeniy Gabrilovich and Alex Gontmakher, "The Homograph Attack", *Communications of the ACM*, 45(2):128, February 2002.

[28] Tobias Holgers, David E. Watson, and Steven D. Gribble, "Cutting through the Confusion: A Measurement Study of Homograph Attacks," in *Proc. USENIX Annual Technical Conference*, June 2006.

[29] DomainSponsor Terms of Use, http://www.domainsponsor.com/terms.html.

[30] Google AdSense for domains, http://www.google.com/domainpark/.

Tracking the Role of Adversaries in Measuring Unwanted Traffic

Mark Allman
ICSI

Paul Barford
University of Wisconsin

Balachander Krishnamurthy & Jia Wang
AT&T Labs–Research

Abstract

Measurements related to security are being carried out on many sites on the Internet at network ingress points, between specific points on the Internet, and across the wide area Internet. The goals range from identifying sources of and possibly filtering unwanted traffic, to characterizing and coming up with new mechanisms for deterring attacks. Most of the measurements do not *systematically* consider adversarial traffic aimed at their measurement system. We explore the role adversaries can play and present a taxonomy on the potential impact of unwanted traffic on measurement systems. Our goal is to both enhance the robustness of such systems and spur development of tools that can alter the playing field by increasing the cost to adversaries.

1 Introduction

Attacks on the Internet are steadily rising and range from identity spoofing, spam, phishing, flooding links, distributed denial of service attacks, worms, virus, to organized targeted attacks by botnets. Various interesting and pertinent solutions have been proposed to reduce the impact of this unwanted traffic—authenticating senders, identifying malware, filtering, throttling, rate limiting, sharing information, etc. Research on unwanted traffic has examined attacks traditionally from the viewpoint of its impact on the user, the end systems, and the network. In addition, numerous measurement projects have characterized malware, examined the breadth of their impact, and helped in the search for solutions.

What has been missing is a *systematic examination* of the specific impact of *adversarial* traffic on the measurement systems that have been created to deal with the malware. Systems to mitigate security threats have been proposed that often also take into account an active adversary trying to evade or otherwise attack them. However, the adversary is considered to be narrowly focused on defeating the mechanism under study. We sketch a framework for a more systematic evaluation that considers the impact an adversary can have on the measurements, and lays the groundwork for considering how these effects can propagate across different types of measurement.

Interference has been anecdotally discussed in certain contexts, such as Web performance measurement systems like Keynote [4] and eValid Test Suite [2]. These systems use a globally distributed set of platforms to monitor customer Web sites by sending periodic requests to selected Web pages to ensure availability and simultaneously measuring latency. Since the location of the monitoring clients is often known, it is not surprising that some Web sites game the system by providing a different (higher) quality of service to requests from such monitoring clients to be perceived as a better provisioned site. This would differ from actual user experience of the site.

Such interference is relatively benign and has impact on competitive businesses. If the systems instead dealt with security related measurements, an active adversary has a strong incentive to use techniques to bypass the system or compromise it. Even if the security infrastructure is not entirely compromised, inferences made without taking into account active adversaries could be flawed. The flaws could be from the perspective of false negatives, false positives, and overall effectiveness.

In this paper we examine the ways in which measurements and thus inferences can be skewed as a result of adversarial actions. Our goal is to offer a framework for which researchers and practitioners can evaluate the impact of adversaries on their measurements from a variety of angles. We first illustrate the adversarial nature of measuring unwanted traffic in § 2. We then present a taxonomy of the impact this traffic has on the measurement systems in § 3. Finally, we conclude with a look towards future work.

2 Measuring Adversaries

In this paper we focus our attention on measurement for security-related tasks. Unlike standard measurement tasks, this type of measurement explicitly involves an *adversary*. That is, network operators are using the measurements to either enforce some given policy on the traffic (*e.g.*, with a firewall or intrusion prevention system) or to better understand the behavior of some remote entity with the idea of informing future detection and response

mechanisms (*e.g.*, with Honeypots). On the other hand, attackers sometimes try quite hard to not be noticed and to circumvent efforts at being measured and characterized. This tension is fundamental to all security-related measurement activities. In this section, we first survey the types of measurement systems used in the Internet today for security-related measurement. We then categorize the types of unwanted traffic that these systems are likely to encounter and the possible impact of this unwanted traffic.

2.1 Measurement Systems

Many schemes have been developed and used to measure and characterize unwanted traffic. We focus on four broad categories of popular measurement systems.

Firewalls: One of the most widely deployed mechanisms for detecting and controlling unwanted traffic is a firewall at the edge of an enterprise network. We consider "firewalls" to be a range of devices and activities. One end of the spectrum are simple Access Control Lists on routers that approve or deny traffic based on per-packet features (*e.g.*, IP addresses, transport layer port numbers, etc.). Firewalls can also be dedicated machines on the network path that keep per-flow state and support complex security policies such as capping the sending rate of a connection or the rate at which a machine can initiate connections. Firewalls also have a range of logging capabilities, from none to quite detailed.

NIDS: A second type of security-related measurement system is a Network Intrusion Detection Systems (NIDS), such as Snort [8] or Bro [6]. NIDS are deployed at strategic points to monitor incoming and outgoing traffic for all the hosts on a network. NIDS can produce both alarms for suspicious activities, as well as interface with firewalls to automatically stop ongoing attacks. NIDS use either signatures or algorithms to separate benign and malicious traffic. For instance, a well-known string (signature) within an HTTP GET request may indicate that an incoming packet is attempting to infect a local Web server with a particular worm. In another case, a NIDS may employ an algorithm to watch connection attempts over time to identify a remote host as a scanner ([6] shows one such algorithm). Another common technique for NIDS to use is to somehow characterize "normal" traffic and then flag deviations from that baseline (so-called "anomaly detection").

Honeypots: The third class of measurement systems we consider are network Honeypots [7, 13]. These systems either are real hosts or mimic real hosts attached to routed but otherwise unused address space. All traffic that arrives at these hosts is presumed to be either the result of a mis-configuration or malicious. Therefore, by actively responding to the queries, the Honeypots can be used to characterize unwanted traffic for the purposes of (*i*) warning operators (and other devices such as firewalls and NIDS) of previously unseen attacks and (*ii*) providing trends that help to improve operators situational awareness [12].

Application-Level Filters: These systems reside at the application layer and attempt to determine whether arriving traffic or requested traffic is "wanted" or not. The two most prevalent kinds of filters are for incoming email and for Web requests routed through a proxy server. Arriving email is often scanned for viruses and spam using a myriad of techniques from matching signatures to statistical techniques based on the prevalence of various terms in the email. Another example is enterprises using Web proxies that filter requests based on the content presumed to be associated with a given URL (*e.g.*, companies barring employees from accessing unsavory Web sites).

2.2 Adversarial Traffic

We next look at the range of traffic types that present challenges for security-related measurement systems. We break this down into three groups: (*i*) traffic that attempts to attack the measurement system or its resources directly, (*ii*) traffic that attempts to evade or circumvent the measurement systems, and (*iii*) traffic that intentionally avoids the measurement systems to prevent characterization.

2.2.1 Direct Attacks

Direct attacks on security-related measurement infrastructure can come in two forms.

First, attacks can simply try to DDoS the systems resources, be it bandwidth, memory, table entries, etc. As an example, some IDS systems such as Bro [6] maintain a large amount of state about various streams of traffic at various layers in the protocol stack simultaneously. One method of attacking such a system would be to attempt to make the system track more traffic streams than it could handle and therefore be able to sneak a malicious flow by the system without the system noticing (and, therefore, generating an alarm) [1]. Alternatively, many IDS systems attempt to buffer out-of-order packet arrivals to construct the exact byte stream an application is given. An attacker could attempt to use such a buffer against the measurement system and fill enough memory that new traffic would not be appropriately monitored. Another form of attack is to slowly ramp up "background noise" (legitimate Web fetches or similar benign traffic) in the victim's network to bring the measurement systems closer to the brink of failing to be able to keep up, with an attack then simply providing the proverbial straw that breaks the entire system.

A second type of direct attack on security-related measurement infrastructure is to compromise the measurement platform itself. For instance, the Witty worm [9] compromised hosts running various Internet Security Systems (ISS) products. Once a security-related measurement system has been compromised the alerts generated or actions taken are clearly questionable at best.

2.2.2 Evasion

We next focus on attacks that attempt to evade the measurement systems. We provide several examples of this sort of attack, but do not claim that this is a comprehensive list.

Among the simplest evasion techniques are splitting up payloads amongst multiple small packets (using IP fragmentation, multiple TCP packets, etc.). A simple measurement system that makes judgments on each arriving packet independently of all other packet arrivals will be easily fooled by splitting some string across packets. For instance, instead of sending "`root`" in one packet it gets broken into two packets with "`ro`" and "`ot`".

Another tactic is to attempt to circumvent a firewall that does not pass traffic from some application by using a non-standard port for the application in question, but one that the firewall does allow. For instance, using TCP port 80 for *ssh* traffic rather than for Web traffic. This can be exploited by benign users that are simply trying to get work done or by adversaries who are trying to hide their traffic from security monitors.

Evasion tactics get trickier. For instance, an attacker may leverage the IP TTL to show a NIDS a superset of the packets that will actually arrive at the end host, thus giving the NIDS a potentially bogus view of what might be happening on the end system [3].

Another trick (from [3]) is to reorder and retransmit segments to try to confuse a NIDS system. For instance, if "`root`" is to be transmitted the attacker might send "`m`" with sequence number 4, "`oo`" with sequence numbers 2 and 3, a "`t`" with sequence number 4 and finally an "`r`" with sequence number 1. This leaves the NIDS with an inconsistent view. Did the attacker send "`root`" or "`room`"? The TCP specification is ambiguous and actual behavior varies. Therefore, the NIDS may get fooled into thinking something benign was transmitted instead of something malicious, or visa-versa.

NIDS systems that use anomaly detection are susceptible to attacks whereby the adversary attempts to re-define the notion of "normal" such that it is easier for an attack to fly "under the radar". A similar situation exists for Honeypots which commonly use statistical methods to interpret data.

Spam filters may represent the single system that attackers most often attempt to circumvent. Basic systems that simply match strings are dealt with by changing the case of text, mis-spelling words, breaking lines at different points, etc. More advanced statistical techniques for finding spam are gamed by adding benign sounding words, mis-spelling words, using HTML formatting tricks to split words up in the actual email, but have those words put back together when rendered for the user, including the spam message in a graphic instead of as text, etc. This is a year's old arms race without any signs of stopping, illustrating the difficulty of the task for measurement systems that attempt to characterize an adversary.

2.2.3 Avoidance Attacks

A final type of attack is an avoidance attack. The idea is that if an attacker can determine that it is interacting with a Honeypot rather than a "real" end system then the attacker may avoid the block of addresses the Honeypot is using. This kind of blacklisting can leave a void in the characterization of some specific malicious activities. This void then may propagate to other measurement systems. For example, IDS systems then will not have the signatures that a Honeypot may have been able to generate. This problem can be addressed by making the Honeypots as realistic as possible and by making the Honeypots monitor a constantly changing set of addresses to defeat blacklisting [5].

3 Modalities for Measurement Pollution by Unwanted Traffic

Our adversarial models imply that Internet measurements can be affected by unwanted traffic in a variety of ways. However, *adversarial intent* alone is insufficient to fully explain the scope of this issue. We posit that a complete description of how unwanted traffic effects measurements must include consideration of the *target measurement system.* For our purpose this means both the "sensor" component of the measurement system that is used to collect the raw data, and the "analysis engine" that is used to transform the raw data into the form that is presented to and considered by users.

For this paper, we consider four systems (explained in the previous section) used for the specific purpose of measuring unwanted traffic—firewalls, NIDS, Honeypots and application-level filters. Unwanted traffic can pollute the results reported by each of these systems in different ways.

We describe a taxonomy of the ways in which unwanted traffic from adversaries can change the measurements generated by firewalls, NIDS, and Honeypots based on two concepts: *consistency* and *isolation.* Both

of these concepts are defined in terms of a set of packets $P = p_1 \ldots p_n$ that arrive at the measurement system, and the resulting log entries available to users $A = a_1 \ldots a_m$. We define a measurement system to be consistent when a given set of packets P_i always results in the same set of log entries A_j. We define isolation in a measurement system if a given set of packets P_i results only in the set of log entries A_j. It is important to emphasize that the log entries A_j for each of the three target measurement systems are different. In the case of NIDS, log entries are alarms that include summary information from the rule that matched a set of packets and some detail on the packets themselves. Entries in firewall logs are similar. In contrast, log entries for Honeypots are often only packet traces that can include either headers alone or headers plus payloads. With these definitions, we enumerate the taxonomy as follows:

Consistent/Isolated This is the case where the measurement system is behaving correctly and log entries are not altered in unexpected ways by unwanted traffic. Specifically, given instances of unwanted traffic $p_i, \ldots, p_t$ generate log entries $a_i, \ldots, a_t$ and no other set of packets P_w will have an impact on the log entries. This ideal, predictable behavior is the baseline against which other cases must be compared.

Consistent/Non-isolated This is the case where the measurement system behaves in a consistent, predictable way, but where a given instance of unwanted traffic changes the log entries caused by other unwanted traffic. Specifically, consider that a given instance of unwanted traffic P_i results in log entry A_i. If another set of unwanted traffic P_j arrives along with P_i then the resulting alarm will change to A_k The following example illustrates a real world instance of this case.

Two sequences of packets were created using the MACE malicious workload generator [10] and submitted to a system running the Snort NIDS version 2.4.4 with rules public release 2.4. Sequence S_1 triggers a pair of alarms (shown in Table 1) because of the string "/hsx.cgi" in the URI (first alarm) and the string "../../" (second alarm) in the payload (either after "?" for a GET request or in the body of a POST). Sequence S_1 consists of three packets: the first for the HTTP request up to the end of the "/hsx.cgi" string and the second and third packets consisting of the payload "../". The second sequence, S_2, differs from the first in that an additional packet was inserted between the second and third packets consisting of "`\x08/`", which is equivalent to a backspace followed by a forward slash. When Snort observes sequence S_2 only the first alarm in Figure 1 is fired, even though it is semantically equivalent to S_1. This example is similar in most respects to common insertion exploits; the difference being that in this case the resulting log entries are changed, not omitted. This problem is similar to the issue of normalization [3], but at a payload semantic level.

Inconsistent/Isolated This is the case where the measurement system generates two (or more) different sets of alarms A_{i1} and A_{i2} when the same set of packets P_i arrives at the system in two separate measurement epochs. The difference between this case and the last is that the sets of alarms A_{iX} are not predictable, and that the alarms generated by other sets of packets P_j are not affected by this behavior. This case can arise when certain kinds of randomness are introduced into the environment that lead to unpredictable behavior.

It is difficult to construct a concise examples for this case since inconsistency in the pure sense should not be bounded over time. However, we offer two examples of inconsistent/isolated behavior that illustrate the idea. The first example is an extension of the example in Table 1. It is likely that there are other signatures for attacks on a particular service that could be affected by the presence of a backspace character in a packet – but unlikely that all signatures in the database would be vulnerable. If a series of these backspace packets were directed toward a NIDS like Snort that considers traffic on a packet-by-packet basis, their impact would be unpredictable and would depend directly on whether the packets arrived in an order sufficient to alter the resulting log entries. This is inconsistent behavior. The fact that only signatures susceptible to the backspace could be affected means that the behavior is isolated.

A second example of this case relates to the idea of packet interleaving described in [11]. For NIDS that are connection-oriented such as Bro, signatures can be expressed in the form of state machines in which multiple packets from a source may be required to raise an alarm. In this case, two different signatures might have common initial sequences. If the same set of packets from a given service (such as NetBIOS) are observed by the NIDS in a different order between two measurement epochs, there is both the possibility that a different order of alarms will be logged and that a different set of alarms will be raised. These kinds of experiments were conducted in [11], and both of these outcomes were observed.

Inconsistent/Non-isolated This final case is where the measurement system's behavior in terms of the resulting logs is truly unpredictable from one epoch to the next. In this case, if a set of packets $p_i, \ldots, p_j$ result in log entries $a_i, \ldots, a_j$ in one measurement epoch, the same set of packets could result in log entries $a_k, \ldots, a_l$ in the next measurement epoch. This case is important because it illustrates that randomness in unwanted packet streams may not only be an issue of arrival order as in the case above, but can also be caused by the entirely different issue of denial of service. As discussed earlier, it is well known that many different measurement

Packet sequence S_1:	payload 1: `POST /hsx.cgi HTTP/1.0\r\nContent-length: 10\r\n\r\n` payload 2: `../` payload 3: `../` payload 4: `\x00\x00\x00\x00`
Packet sequence S_2:	payload 1: `POST /hsx.cgi HTTP/1.0\\r\\nContent-length: 10\r\n\r\n` payload 2: `../` payload 3: `\x08/` payload 4: `../` payload 5: `\x00\x00\x00\x00`
Snort rule #1:	uricontent:"/hsx.cgi"; (*i.e.,* if /hsx.cgi appears in the URI, raise an alarm)
Snort rule #2:	uricontent:"/hsx.cgi"; content:"../../"; content:"%00"; distance:1; (*i.e.,* if /hsx.cgi appears in the URI, plus content of ../.. followed by null byte in payload, raise an alarm)
Alarm #1:	[**] [1:1113:5] WEB-MISC http directory traversal [**] [Classification: Attempted Information Leak] [Priority: 2] 04/18-18:46:45.587011 10.2.23.103:25868 => 10.2.0.2:80 TCP TTL:64 TOS:0x0 ID:41059 IpLen:20 DgmLen:58 DF ***AP*** Seq: 0xECAE373E Ack: 0x43344649 Win: 0x8218 TcpLen: 32 TCP Options (3) => NOP NOP TS: 633549769 553538167 [Xref => http://www.whitehats.com/info/IDS297]
Alarm #2:	[**] [119:18:1] (http_inspect) WEBROOT DIRECTORY TRAVERSAL [**] [Classification: Attempted Information Leak] [Priority: 2] 04/18-18:46:45.587011 10.2.23.103:25868 => 10.2.0.2:80 TCP TTL:64 TOS:0x0 ID:41059 IpLen:20 DgmLen:58 DF ***AP*** Seq: 0xECAE373E Ack: 0x43344649 Win: 0x8218 TcpLen: 32 TCP Options (3) => NOP NOP TS: 633549769 553538167 [Xref => http://www.whitehats.com/info/IDS297]

Table 1: Snort generates both alarm#1 and #2 when it receives packet sequence S_1. However, Snort only generates alarm #1 when it receives packet sequence S_2 which includes a back space character in the fourth packet. This is an example of consistent, non-isolated behavior.

systems are susceptible to DoS attacks including NIDS and firewalls [10]. DoS attacks can cause unpredictable resource consumption in these systems leading to unpredictable packet loss and other effects that have the potential to alter what is ultimately logged by the measurement system.

4 Conclusions and Future Work

We have made an initial examination of how adversaries could impact security related measurements that are ongoing at various protocol layers in numerous sites across the Internet. We have sought to enumerate adversarial traffic types and to provide a taxonomy of the impact of this traffic on security related measurements. Our next steps in this study include conducting experiments with a spectrum of unwanted traffic and measurement systems to assess the actual impact in a controlled environment. Our ultimate goal is to develop generic tools and practices that can augment the ability of measurement systems to be robust against unwanted traffic.

Acknowledgments

Mark Allman's work was supported in part by the National Science Foundation under grants ITR/ANI-0205519, NSF-0433702 and STI-0334088. Paul Barford's work was supported in part by the National Science Foundation grants CNS-0347252, CCR-0325653. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the above government agencies or the U.S. Government.

References

[1] S. Crosby and D. Wallach. Denial of Service via Algorithmic Complexity Attacks. In *USENIX Security Symposium*, 2003.

[2] eValid WebSite Analysis and Testing Suite. http://www.soft.com/eValid/.

[3] M. Handley, C. Kreibich, and V. Paxson. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *Proc. of the 10th USENIX Security Symposium*, August 2001.

[4] Keynote. http://keynote.com.

[5] B. Krishnamurthy. Mohonk: Mobile Honeypots to Trace Unwanted Traffic Early. In *Proc. of the ACM SIGCOMM Network Troubleshooting Workshop*, 2004.

[6] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proc. of the 7th USENIX Security Symposium*, January 1998.

[7] N. Provos. A Virtual Honeypot Framework. In *Proc. of the 13th USENIX Security Symposium*, August 2004.

[8] M. Roesch. Snort - Lighweight Intrusion Detection for Networks. In *Proc. of the 13th USENIX Systems Administration Conference*, November 1999.

[9] C. Shannon and D. Moore. The Spread of the Witty Worm. *IEEE Security and Privacy*, 2(4):46–50, Aug. 2004.

[10] J. Sommers, V. Yegneswaran, and P. Barford. A Framework for Malicious Workload Generation. In *Proc. of the ACM Internet Measurement Conference*, October 2004.

[11] J. Sommers, V. Yegneswaran, and P. Barford. Recent Advances in Network Intrusion Detection Systems Tuning. In *Proc. of the 40th IEEE Conference on Information Sciences and Systesm*, March 2006.

[12] V. Yegneswaran, P. Barford, and V. Paxson. Using Honeynets for Internet Situational Awareness. In *Proc. of the ACM/USENIX Fourth Workshop on Hot Topics in Networks*, November 2005.

[13] V. Yegneswaran, P. Barford, and D. Plonka. On the Design and Use of Internet Sinks for Network Abuse Monitoring. In *Proc. of the Symposium on Recent Advances in Intrusion Detection*, September 2004.

An Algorithm for Anomaly-based Botnet Detection

James R. Binkley
Computer Science Dept.
Portland State University
Portland, OR, USA
jrb@cs.pdx.edu

Suresh Singh
Computer Science Dept.
Portland State University
Portland, OR, USA
singh@cs.pdx.edu

Abstract

We present an anomaly-based algorithm for detecting IRC-based botnet meshes. The algorithm combines an IRC mesh detection component with a TCP scan detection heuristic called the *TCP work weight*. The IRC component produces two tuples, one for determining the IRC mesh based on IP channel names, and a sub-tuple which collects statistics (including the TCP work weight) on individual IRC hosts in channels. We sort the channels by the number of scanners producing a sorted list of potential botnets. This algorithm has been deployed in PSU's DMZ for over a year and has proven effective in reducing the number of botnet clients.

1 Introduction

Botnets [6] [5] are a current scourge of the Internet. Botnets may result in high rates of TCP syn scanning (for example, see [4]), voluminous spam, or distributed DOS attacks (see [2]).

At Portland State University, in the last few years we began to realize that many of our security incidents had a common thread which proved to be botnet related. As a result we developed an anomaly-based algorithm for detection of botnet client meshes and made it a subcomponent of our open-source ourmon [3] [9] network management and anomaly detection system. The system is currently deployed in our DMZ where we see peak traffic periods of 60k pps. In the last year, this system has proven beneficial in reducing the number of botnet clients on campus.

Our anomaly-based system combines an IRC [7] parsing component with a syn-scanner detection system aimed at individual IP hosts. The IRC parsing system collects information on TCP packets and determines an IRC channel, which we define as a set of IP hosts. We then correlate the IP host information over a large set of data sampled during the current day which tells us if an individual host in the IP channel was a scanner. We then sort the IRC channels by scanning count, with the top suspect channels labeled as possible *evil channels*. This algorithm is not signature-based in any way. It does not rely on ports or known botnet command strings. As a result, we are immune to zero-day problems. Our algorithm does assume that IRC is cleartext and that attacks are being made with the botnet mesh.

2 IRC Botnet Detection Algorithm

Our architecture relies on the observation that IRC hosts are grouped into channels by a channel name (for example, "F7", or "ubuntu" might be channel names), and that an evil channel is an IRC channel with a majority of hosts performing TCP SYN scanning.

The front-end data collector gathers three kinds of list tuples that are useful for benign IRC, botnet detection, and scanner detection. The tuples consist of two kinds of IRC tuples and the TCP syn tuple. The probe gathers these tuples over its thirty second sampling period. All tuples are then sent to the back-end for further processing. In the probe, the entire campus TCP syn tuple set is filtered into a smaller subset which informally consists of hosts observed sending anomalous amounts of TCP syns. This syn tuple subset is called the "worm set" and is typically orders of magnitude smaller than the entire set of IP sources found in the campus TCP syn set.

The TCP syn scanner list tuple has the following simplified form:

```
(IP source address, SYNS, SYNACKS,
FINSSENT, FINSBACK, RESETS,
PKTSSENT, PKTSBACK)
```

The logical key in this tuple is an IP source address. SYNS, FINS (all kinds), and RESETS are counts of TCP control packets. SYNS are counts of SYN packets sent from the IP source, and SYNACKS are a subset of only

those SYNS sent with the ACK flag set. FINS sent both ways are counted. RESETS are counted when sent back to the IP source. The PKTSSENT counts the total packets sent by the IP source. PKTSBACK counts the total pkts returned to the IP source. Other fields exist but are not relevant to this paper. This information is useful for determining what kind of scanning is occurring and often gives a rough network-based indication of the kind of exploit in use.

We define a metric which we call the *TCP work weight*. The *work weight* is easy to compute and is computed by the probe per IP source as follows:

$$w = (S_s + F_s + R_r)/T_{sr}$$

It is expressed as a percent. The rough idea is that we take the count of TCP control packets (SYN's plus SYNACKs sent, FIN's sent and RESETS) and divide that count by the total number of TCP packets (T_{sr}). Obviously 100% here is a bad sign and implies a true anomaly of some sort. Such a value is typically associated with a scanner or worm although some forms of P2P (and email servers) may have high work weights for shorter periods of time. The IRC module in the probe uses the TCP list as an underlying "tool", and extracts the TCP work weight from it for any IRC host.

We should point out that we have over two years worth of experience with the work weight at this point. We have learned that high work weights with hosts are caused by three possible causes including 1. scanners (typically syn scanners), 2. clients lacking a server for some reason or 3. P2P hosts (usually Gnutella is the application) IP peers. In general scanners are the most common reason for a high work weight. We also know that that the work weight clusters into either high values or low values (say 0..30%). Attackers fall into the higher range. P2P clients on average fall into the lower range.

Typically the average over many samples is of interest. However in the case of IRC we decided to simply take the maximum work weight seen over all the thirty second samples for a day. This is because an otherwise normal host may be ordered remotely to do scanning for a short period of the day. One host by itself in an IRC channel with a high work weight may not be anomalous. However if a channel has ten hosts out of twelve with high work weights suspicion is justified. As a result, work weights associated with IRC channels in our summarization reports are maximum weights seen across all the samples in a daily report.

There are two IRC lists, called the *channel list* and the *node list*. The channel list has the following tuple structure:

```
(CHANNAME, HITS, JOINS, PRIVMSGS,
NOIPS, IP_LIST}
```

The channel name is the case-insensitive IRC channel name extracted from JOIN and PRIVMSG IRC messages by the IRC scanner. The probe's scanner is handcrafted C code that looks at the first 256 bytes of the L7 payload for TCP messages only and extracts IRC tokens for the four kinds of messages of interest. HITS is the total count of JOINS and PRIVMSGS, JOINS and PRIVMSGS are counts of that particular kind of message. NOIPS is the number of IP addresses in the IP_LIST, which follows the tuple. Thus a channel tuple gives a key (the channel name) with a few message count statistics and a list of IRC hosts in the channel expressed as IP addresses.

The *node list* gives per IP statistics for any IP address in any IRC channel. Informally a channel may be viewed as a directory, and a host may be viewed as a directory entry (although a host may actually be in more than one channel). The node list has the following tuple structure (not all counters shown):

```
(IPSRC, TOTALMSG, JOINS, PINGS, PONGS,
PRIVMSGS, CHANNELS, SERVERHITS, WW}
```

The key per tuple is an IP source address. Various message statistics are given including JOIN, PING, PONG, and PRIVMSG counts. The number of observed per host channels is supplied. SERVERHITS indicates the number of messages sent to/from a host. Thus this counter indicates whether a host is acting as an IRC server. The WW (work weight) as mentioned previously is derived from the TCP syn module.

One additional IRC statistic is gathered by the front-end which consists of total counts of the four kinds of IRC messages seen by the probe during the sample period. (This tuple is displayed by the back-end as an RRDTOOL-based graph - due to space limitations we cannot show such a graph in the paper). It shows that IRC is basically a slow phenomenon with only a few messages per second, even though our campus may have 5000 IP hosts active during a day. As a result our IRC evil channel analysis is based on a slower time scale, hours and days.

The IRC tuples are passed to the backend for report generation. The backend program produces an hourly text report (updated on the hour) which is called *ircreport_today.txt*. This file is available on the web for analysis. Data in this report is broken up into three major sections including global counts, channel statistics, and per host statistics. Channel statistics and per host statistical sections are further broken up into various sub-reports where data is typically sorted by some key statistic.

We can distinguish the following IRC report subsections:

1. evil channels - channels with too many hosts with a high work weight

2. channels sorted by maximum messages.

3. channels with host statistics - each channel shows the host IP in the channel with host stats.

4. servers sorted by max messages - hosts that are IRC servers are sorted by max messages.

5. hosts with join messages but no privmsgs - JOINs only but no data payloads.

6. hosts with any signs of worminess - hosts with high work weights.

For purposes of illustration in table 1 we look at one benign example which comes from the per channel host statistics section. Counts given for our example were taken from twelve hours of data (since midnight) and are typical for a small IRC chat group. [1]

In our example 1, a channel named "ubuntu" has four hosts in it. Three are local and the server (S) is remote. Total message counts (of the 4 kinds parsed) and JOIN, PING, PONG, and PRIVMSG counts are given. Maxchans is the number of channels seen during the period for that host, and maxworm is the maximum work weight seen. We do not believe this channel based on the above data is "evil".

Now let us see how this data may be correlated to plainly point out anomalous IRC-based botnet behavior.

3 Botnet Examples

Let us look at three examples which illustrate the operation of our algorithm. Table 2 gives us three items from our evil channel report. The purpose of this selection is to illustrate, on the one hand, the effectiveness of our algorithm in detecting evil channels while on the other hand showing some borderline cases that require additional analysis (e.g., examining port reports) First, we present a botnet client mesh. By definition, the server is off-campus and a few hosts have been captured on-campus to become part of the botnet. We look at two subsections of the hourly IRC report to find our evil channel which is named "F7". We look at our evil channel sort, and discover that F7 shown in table 2 is named as a channel in that list and occupies a high rank in the list.

Channel F7 is high in the evil channel list simply because it has 4 out of 6 hosts with high work weights. The "evil" flag at the end of the column is set to E if a potential evil channel has more than 1 anomalous host. Next we look at the report sub-section which breaks host statistics out for the channel F7.

[1] All IP addresses have been changed and are represented as symbolic addresses. In addition the reader should note that our current output format is simply an ASCII report. However we represent it here in tabular format.

In table 3 we see the part of the report that shows hosts in a channel. In channel F7, we have one remote server and five infected local hosts. Four of those hosts have very high maximum work weights. We know from experience with the work weight (and also by looking at logs from both Ourmon and other systems) that the hosts are performing SYN scanning. Ourmon logs for the syn tuple will typically show that the hosts in question have been performing scanning aimed at Microsoft exploits on port 445 (typically lsass-based exploits, for example, see [4]).

We have used ngrep in the past to prove beyond a shadow of a doubt that examples like our F7 botnet client are indeed malign. At this point in time, we no longer feel the need to use a tool like ngrep to prove that ourmon has detected an evil mesh. However the reader might desire to see such proof and in addition ngrep can still be very useful as an aid in host forensics. For example, one may be able to gather valuable clues about the exploit used. An ngrep sent from a local client to the server in question (net2.1) showed messages like the following:

```
# ngrep -q host net2.1
T net1.1:1053 -> net2.1:30591 [AP]^
  PRIVMSG #F7 :[Lsass]: Fuxed IP: net1.2
```

Here we see a report from a bot client back to the server that host net1.2 has been exploited. The exploit used is also mentioned.

The other two examples in table 2 are not evil channels. s3reporter is a IRC game (which is why all participating IP addresses are marked as servers) for which we sometimes get a high work weight. However, since the high work weight is associated with a remote host (see table 3), we do not consider it further. The third example has a local IP host with a high work weight which implies evil channel. However, this is a borderline case (with only one client) where the high work weight may be because of software glitches (e.g., meetingmaker loss of server causes this type of bot-like behavior) or a p2p outage of some sort. These types of channels require additional analysis where we need to examine port reports in more detail. Some of the specifics that we look for include, for example,

- L3/L4 dst counts of unique L3 and L4 destinations. This can suggest whether a host is traversing IP destinations or ports or both.

- EWORM - a flag system to indicate if traffic exists that is 2-way or if network errors exist. E.g., E and R indicate ICMP errors or RESETS returned to the host. O indicates a lack of FINS. M indicates no non-control packets returned.

Table 1: Benign IRC Channel - Channel/Host Report

channel/ip	tmsg	tjoin	tping	tpong	tprivmsg	maxchans	maxworm	server
ubuntu/net1.host1	11598	1282	1912	1910	6494	4	43	H
ubuntu/net1.host2	7265	938	619	622	5086	3	0	H
ubuntu/net1.host3	17218	1926	4123	4100	7069	5	37	H
ubuntu/net2.host1	28152	3222	3913	3904	17113	8	0	S

Table 2: Malign and normal IRC Client Botnet - Evil Channel Report

channel	msgs	joins	privmsgs	ipcount	wormyhosts	evil
F7	118	19	99	6	4	E
s3reporter	2259	25	2234	3	1	E
thespicebox	23	8	15	2	1	E

Table 3: Malign and Benign Channels - Channel/Host Report

channel/ip	tmsg	tjoin	tping	tpong	tprivmsg	maxchans	maxworm	server
F7/net1.1	1205	24	377	376	428	2	42	H
F7/net1.2	113	6	39	43	25	1	96	H
F7/net1.3	144	2	60	61	21	1	94	H
F7/net1.4	46	3	12	14	17	1	90	H
F7/net1.5	701	2	343	345	11	1	90	H
F7/net2.1	1300	19	587	593	101	1	16	S
s3reporter/net1.1	3949	25	844	846	2234	1	5	S
s3reporter/net2.1	6899	36	794	794	5275	2	90	S
s3reporter/net3.1	4525	21	704	702	3098	2	19	S
thespicebox/net1.1	3106	101	433	661	1911	2	83	H
thespicebox/net2.1	10943	373	1828	2037	6705	4	43	S

- Sampled destination ports sometimes help to characterize the nature of the attack. For example, given that in our network we know that Microsoft file share ports are blocked, scanning of port 139 and 445 is deeply suspicious. In addition nearly all of our bot clients caught in the last year were scanning those ports.

Thus, in the end, while out algorithm clearly shows the presence of evail botnets, for many boderline cases, we need to resort to additional analysis.

4 Related Work

In general the academic literature on botnet detection is sparse. Furthermore we are not aware of any other anomaly-based system for detection of botnets. Known techniques include honeynets and IDS systems with signature detection. Honeynets [6] or darknets might be distributed [1] or local and can certainly prove beneficial in terms of providing information about botnet technology. However they may not be easily deployed in a commercial environment and do not necessarily help with the question of whether host X has worm Y. Knowledge of useful signatures and behavior of existing botnet systems is another venue for detection. The paper [2] presents a good introduction to botnets and analyzes botnet architecture. An open-source system like snort [8] can be used for detection of known botnets.

The problem with signatures is of course one may lack the required signature for a bot known elsewhere, or a bot may be new to the world, locally unknown, or changed, thus defeating previously known signatures. Anomaly detection on the other hand may detect such a system. Problems with anomaly detection can include detection of an IRC network that may be a botnet but has not been used yet for attacks, hence there are no anomalies. As our technology depends on hackers actually launching attacks, there is no guarantee that we can detect every infected system. One can also argue that anomaly detection is "too late". It is certainly better to detect an initial attack with a signature when it first occurs and get an exploited system fixed before it is used for spam or denial of service attacks. We believe signatures and anomaly detection are often complimentary and should not be viewed as somehow competitive. All of these techniques (honeypot, IDS, and anomaly detection) can be useful and provide slightly different set of information.

5 Conclusion

In this paper we have presented our current system for bot anomaly detection. We discussed how we combined TCP-based anomaly detection with IRC tokenization and IRC message statistics to create a system that can clearly detect client botnets and how also gross statistical measures can easily reveal bot servers. This system is currently deployed in our network and works well.

The white paper [5] calls for systems to detect botnets via more robust detection means. We believe our current anomaly-based detection system is an advance in the art, but it could be easily defeated by simply using a trivial cipher to encode the IRC commands. As a result we would lose information about mesh connectivity. On the other hand we believe that detection and correlation of attacking meshes of hosts is a valuable contribution. Our current system should be made more general in terms of attacks and include email and DOS attack indicators. For future work, we intend to pursue an anamoly-based algorithm that will work only with layer 3 and layer 4 statistics.

References

[1] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, The Internet Motion Sensor: A Distributed Blackhole Monitoring System. *In Proceedings of the Network and Distributed Security Symposium*, San Diego, CA, January 2005.

[2] P. Barford, V. Yegneswaran, An Inside Look at Botnets, *Special Workshop on Malware Detection, Advances in Information Security*, Springer Verlag, 2006

[3] J. Binkley, B. Massey, Ourmon and Network Monitoring Performance. *Proceedings of the Spring 2005 USENIX Conference, Freenix track*, Anaheim, April 2005.

[4] CERT Advisory CIAD-2004-10 Multiple Vulnerabilities in Microsoft Products `http://www.cert.org/advisories/ciad-2004-10.htm`, April 2004.

[5] E. Cooke, F. Jahanian, and D. McPherson, The zombie roundup: Understanding, detecting and disrupting botnets. *In Proceedings of Usenix Workshop on Stepts to Reducing Unwanted Traffic on the Internet (SRUTI '05)*, Cambridge, MA, July 2005.

[6] The Honeynet Project and Research Alliance. Know Your Enemy, Tracking Botnets. `http://honeynet.org/papers/bots`, March 2005.

[7] J. Oikarinen, D. Reed. Internet Relay Chat Protocol. IETF RFC 1459, May 1993.

[8] Snort IDS web page. `http://www.snort.org`, March 2006.

[9] Sourceforge Ourmon web page. `http://ourmon.sourceforge.net`, December 2005.

[10] Wikipedia web page. `http://en.wikipedia.org/wiki/Internet_Relay_Chat`, December 2005.

Revealing Botnet Membership Using DNSBL Counter-Intelligence

Anirudh Ramachandran, Nick Feamster and David Dagon
College of Computing, Georgia Institute of Technology
{avr, feamster, dagon}@cc.gatech.edu

ABSTRACT

Botnets—networks of (typically compromised) machines—are often used for nefarious activities (*e.g.*, spam, click fraud, denial-of-service attacks, etc.). Identifying members of botnets could help stem these attacks, but *passively* detecting botnet membership (*i.e.*, without disrupting the operation of the botnet) proves to be difficult. This paper studies the effectiveness of monitoring lookups to a DNS-based blackhole list (DNSBL) to expose botnet membership.

We perform *counter-intelligence* based on the insight that botmasters themselves perform DNSBL lookups to determine whether their spamming bots are blacklisted. Using heuristics to identify which DNSBL lookups are perpetrated by a botmaster performing such reconnaissance, we are able to compile a list of likely bots. This paper studies the prevalence of DNSBL reconnaissance observed at a mirror of a well-known blacklist for a 45-day period, identifies the means by which botmasters are performing reconnaissance, and suggests the possibility of using counter-intelligence to discover likely bots. We find that bots are performing reconnaissance on behalf of other bots. Based on this finding, we suggest counter-intelligence techniques that may be useful for early bot detection.

1. Introduction

Internet malice has evolved from pranks conceived and executed by amateur hackers to a global business involving significant monetary gains for the perpetrators [19]. Examples include: (1) unsolicited commercial email ("spam"), which threatens to render email useless by immensely decreasing the signal-to-noise ratio of traffic [17]; (2) denial of service attacks, which have become common [12], and (3) click fraud, whereby a group of attackers send bogus "clicks" for online advertisements that mimic legitimate request patterns, swindling advertisers out of large sums of money [4].

Botnets are a root cause of these problems [8], since they allow attackers to distribute tasks over thousands of hosts distributed across the Internet. A botnet is network of compromised hosts ("bots") connected to the Internet under the control of a single entity ("botmaster", "controller", or *command and control*) [5]. The large cumulative bandwidth and relatively untraceable nature of spam from bots makes botnets an attractive choice for large-scale spamming. Previous work provides further background on botnets [5, 6].

If network operators and system administrators could reliably determine whether a host is a member of a botnet, they could take appropriate steps towards mitigating the attacks they perpetrate. Although previous work has described an *active* detection technique using DNS hijacking technique and social engineering [6], there are few efficient methods to *passively* detect and identify bots (*i.e.*, without disrupting the operation of the botnet). Indeed, detecting botnets proves to be very challenging: a victim of a botnet attack can typically only observe the attack from a single network, from which point the attack traffic may closely resemble the traffic of legitimate users. Regrettably, the state-of-the-art in botnet identification is based on user complaints, localized honeypots and intrusion detection systems, or through the complex correlation of data collected through darknets [13].

We propose a set of techniques to identify botnets using *passive* analysis of DNS-based blackhole list (DNSBL) lookup traffic. Many Internet Service Providers (ISPs) and enterprise networks use DNSBLs to track IP addresses that originate spam, so that future emails sent from these IP addresses can be rejected. For the same reason, botmasters are known to sell "clean" bots (*i.e.*, not listed in any DNSBL) at a premium. This paper addresses the possibility of performing *counter-intelligence* to help us discover identities of bots, based on the insight that *botmasters themselves must perform "reconnaissance" lookups to determine their bots' blacklist status.*

The contributions of this paper include:

1. Passive heuristics for counter-intelligence. We develop heuristics to distinguish DNSBL reconnaissance queries for a botnet from legitimate DNSBL traffic (either offline or in real-time), to identify likely bots. These heuristics are based on an enumeration of possible lookup techniques that botmasters are likely to use to perform reconnaissance, which we detail in Section 2. Unlike previous detection schemes, our techniques are *covert* and do not disrupt the botnet's activity.

2. Study of DNSBL reconnaissance techniques. We study the prevalence of DNSBL reconnaissance by analyzing logs from a mirror of a well-known blackhole list for a 45-day period from November 17, 2005 to December 31, 2005. Section 4 discusses the prevalence of the different types of reconnaissance techniques that

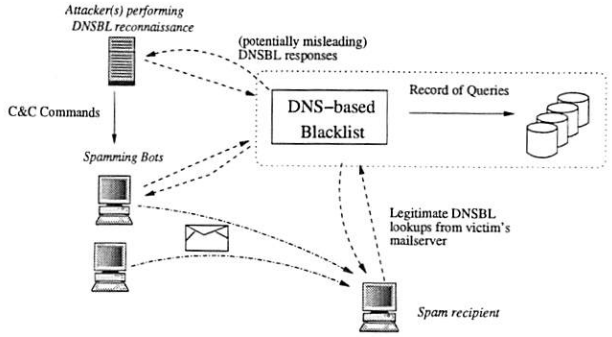

Figure 1: DNSBL-based Spam Mitigation Architecture.

we observed. Much to our surprise, we find that bots are performing reconnaissance on behalf of other (possibly newly infected) bots. Although some bots perform a large number of reconnaissance queries, it appears that much of the reconnaissance activity is spread across many bots each of which issue few queries, thus making detection more difficult.

3. Identification of new bots. We analyze DNSBL queries that are likely being performed by botmasters to identify "clean" bots. Such reconnaissance usually precedes the use of bots in an attack, suggesting the possibility that this DNSBL counter-intelligence can be used to bolster responses. Section 3 demonstrates the possibility of such early warning. To validate our detection scheme, we correlate the IP addresses of these likely bots with data collected at a botnet sinkhole (sinkholing technique explained in previous work [6]) over the same time period (this dataset has been used as "ground truth" for botnet membership in previous studies [6, 17]).

4. DNSBL-based countermeasures. Our heuristics could be used to detect reconnaissance in real-time. This ability potentially allows for active countermeasures, such as returning misleading responses to reconnaissance lookups, as shown in Figure 1. We revisit this topic in Section 5.

2. Model of Reconnaissance Techniques

This section describes our model for DNSBL reconnaissance techniques (*i.e.*, the techniques that botmasters may be using to determine whether bots have been blacklisted). Our goal in developing these models and heuristics is to distinguish DNSBL queries issued by botmasters from those performed by legitimate mail servers.[1]

2.1 Properties of Reconnaissance Queries

Our detection heuristics are based on the construction of a *DNSBL query graph*, where an edge in the graph from node A to node B indicates that node A has issued a query to a DNSBL to determine whether node B is listed. After constructing this graph, we develop detection heuristics based on the expected *spatial* and *temporal* characteristics of legitimate lookups versus reconnaissance-based lookups. These characteristics hold primarily in cases when members of the botnet are not performing queries on behalf of each other, a case that makes detecting reconnaissance more difficult, as we explain in Section 2.2.3. As we describe below, our detection heuristics exploit both spatial and temporal properties of the DNSBL query graph.

Property 1 (Spatial relationships) *A legitimate mail server will perform queries* and *be the object of queries. In contrast, hosts performing reconnaissance-based lookups will only perform queries; they will* not *be queried by other hosts.*[2]

In other words, legitimate mail servers are likely to be queried by other mail servers that are receiving mail from that server. On the other hand, a host that is not itself being looked up by any other mail servers is, in all likelihood, not a mail server. We can use this observation to identify hosts that are likely performing reconnaissance: lookups from hosts that have a high *out-degree* in the DNSBL query graph (*i.e.*, hosts that are performing many lookups) but have a low *in-degree* are likely unrelated to the delivery of legitimate mail. To quantify this effect, we define the *lookup ratio*, λ, of some node n as follows:

$$\lambda_n = \frac{d_{n,out}}{d_{n,in}}$$

where d_{out} is the number of distinct IP addresses that node n queries, and d_{in} is the number of distinct IP addresses that issue a query for node n.[3] This metric is most effective when hosts performing reconnaissance are disjoint from hosts that are actually used to spam, which appears to the case today.However, as reconnaissance techniques become increasingly more sophisticated (as we describe in Section 2.2.3), this metric may become less useful. Still, we find that this metric proves to be quite useful in detecting many instances of DNSBL-based reconnaissance.

The *temporal arrival pattern* of queries at the DNSBL by hosts performing reconnaissance may differ from temporal characteristics of queries performed by legitimate hosts. We expect this to be the case because, whereas legitimate DNSBL lookups are driven by the arrival of actual email, reconnaissance queries will not reflect any realistic arrival patterns of actual email.

Property 2 (Temporal relationships) *A legitimate mail server's DNSBL lookups reflect actual arrival patterns of real email messages: legitimate lookups are typically driven automatically when emails arrive at the mail server and will thus arrive at a rate that mirrors the arrival rates of emails. Reconnaissance-based lookups, on the other hand, will not mirror the arrival patterns of legitimate email.*

We may be able to exploit the fact that email traffic tends to be diurnal [9] to tease apart DNSBL lookups that are

driven by actual mail arrival from those that are driven by reconnaissance. Discovering reconnaissance activity using this method is a topic for future work.

2.2 Reconnaissance Techniques

In this section, we describe three classes of DNSBL reconnaissance techniques that may be performed by botmasters: *single-host, or third-party, reconnaissance*; *self-reconnaissance*; and *reconnaissance using other bots*. For each case, we describe the basic mechanism, the heuristics that we can use to detect reconnaissance in each of these cases, and how each technique may complicate detection.

2.2.1 Third-party Reconnaissance

In *third-party reconnaissance*, a botmaster performs DNSBL lookups from a single host for a list of spamming bots; this host may be the command-and-control of the botnet, or it might be some other dedicated machine. In any case, we hypothesize that the machine performing the lookups in these cases is not likely to be a mail server. Single-host reconnaissance, if performed by a machine other than a mail server, is easily detected, because the node performing reconnaissance will have a high value of λ_n.

Once detected, single-host reconnaissance may provide useful information to aid us in revealing botnet membership. First, once we have identified a single host performing such lookups, the operator of the DNSBL can monitor the lookups issued by that host over time to track the identity of hosts that are likely bots. If the identity of this querying host is relatively static (*i.e.*, if its IP address does not change over time, or if it changes slowly enough so that its movements can be tracked in real-time), the DNSBL operator could take active countermeasures, such as intentionally returning incorrect information about bots' status in the blacklist, a possibility we discuss in more detail in Section 5.

2.2.2 Self-Reconnaissance

Single-host reconnaissance is simple, but it is susceptible to detection. To remain more stealthy, and to distribute the workload of performing DNSBL reconnaissance, botmasters may begin to distribute these lookups *across the botnet itself*. A simple (albeit sub-optimal) way to distribute these queries is to have a bot perform reconnaissance on its own behalf ("self-reconnaissance"); in other words, each bot could issue a DNSBL query to itself (*i.e.*, to determine whether it was listed) before sending spam to the victim.

In this case, identifying a reconnaissance-based DNSBL query is fairly straightforward, because, except in cases of misconfiguration, a legitimate mail server is unlikely to issue a DNSBL lookup for itself. Even though this technique has the advantage of distributing the load of reconnaissance across the botnet, we did not observe this technique being used in practice, likely because a self-query is a dead giveaway.

2.2.3 Distributed Reconnaissance

A more stealthy way to distribute the operation across the botnet is to have each bot perform reconnaissance on behalf of other bots either in the same botnet or in other botnets. For instance, note that Property 1 is unlikely to hold: in this case, the nodes performing reconnaissance will also be queried by other mail servers to which they send spam. As a result, these nodes are likely to have a high $d_{n,in}$, unlike nodes performing single-host reconnaissance. Ultimately, detecting this type of reconnaissance activity may require mining temporal properties (*e.g.*, Property 2).

Although using the botnet itself for DNSBL reconnaissance is more discreet than performing this reconnaissance from a single host, a network operator who positively identifies a small number of bots (*e.g.*, starting with a small hit-list of known bots, probably by using a honeynet with known infected machines). As discussed in Section 4, if this *seed list* of bots performs queries for other hosts, it is likely that these machines are also bots.

We suspected that this mode of reconnaissance would be uncommon, possibly because of the complexity involved in implementing and operating such a system (*e.g.*, keeping track of nodes in the looked-up botnet, disseminating this information to the querying nodes etc.). Much to our surprise, we did witness this behavior; we present these results in Section 4.

3. Data and Analysis

This section describes our data collection and analysis. We first describe our DNSBL dataset and its limitations. Then, we describe how this dataset is used to construct the DNSBL query graph described in Section 2.

3.1 Data Collection and Processing

Our study primarily involves two datasets collected from the same time period (November 17, 2005 to December 31, 2005): (1) the DNSBL query logs to a mirror of a large DNSBL, and (2) the logs of bot connections to a sinkhole for a Bobax botnet [2]. Unlike most botnets, the Bobax bot is designed solely for spamming [1], increasing the likelihood that a query for known Bobax host is the consequence of the querying mail server having received spam from that host.

To verify whether the scheme we propose is indeed able to discover *additional* bots, we compared the IP addresses in the DNSBL query graph against the IP addresses of spammers in a large spam corpus collected at a spam honeypot (the setup of this honeypot is described in our earlier work [17]).

3.2 Analysis and Detection

In this section, we describe how the DNSBL query graph is constructed. Definitions for the terminology used in our algorithm follow: (1) B, the set of IP addresses that attempted to connect to the Bobax sinkhole during the observation period (November 17, 2005–

```
CONSTRUCTGRAPH()
create empty directed graph G

/* Parsing */
for each DNSBL query:
    Identify querier and queried

    /* Pruning */
    if querier ∈ B or queried ∈ B then
        add querier and queried to G if they
            are not already members of G
        if there exists an edge E(querier, queried) ∈ G then
            increment the weight of E(querier, queried)
        else
            add E(querier, queried) to G with weight 1
```

Figure 2: Algorithm to construct a DNSBL query graph

December 31, 2005); (2) *querier*, the IP address of the host that performs a given DNSBL query; (3) *queried*, the IP address of the host that is looked up in a DNSBL query; and (4) G, the DNSBL query graph constructed as a result of the algorithm.

The graph construction algorithm takes as input a set of DNSBL query logs (we use `tcpdump` for packet captures) and the set B and outputs a directed graph G. The algorithm, summarized in Figure 2, consists of two main steps: *parsing* and *pruning*. As the algorithm suggests, we prune DNSBL queries to only include edges which have at least one end (either *querier* or *queried*) present in the set B. Pruning is performed for efficiency reasons: the full DNSBL query logs mostly contain queries from legitimate mail servers. Using B to prune the complete query graph allows us to concentrate on a subgraph which has a higher percentage of reconnaissance lookups than the unpruned graph. We recognize that our analysis will overlook reconnaissance activity where both the *querier* or *queried* nodes are not members of B. To address this shortcoming, we perform a *query graph extrapolation* after the algorithm is run. In this step, we make a second pass over the DNSBL query logs and add edges if at least one of the endpoints of the edge (*i.e.*, either *querier* or *queried*) is already present in the graph. Query graph extrapolation is repeated until no new edges are added to G.

We then compute λ_n for each node in the graph (Property 1), which allows us to identify nodes involved in reconnaissance techniques described in Section 2. Although the results in Section 4 suggest that some bots have large values of λ_n, techniques that use a large number bots to look each other up may be undetectable with this metric. We are developing techniques based on Property 2 to further improve our detection.

4. Preliminary Results

This section presents preliminary results using Property 1 to identify DNSBL reconnaissance activity on the observed DNSBL query graph. We emphasize that the reconnaissance being performed by bots is distinctly under the radar as far as total DNSBL traffic is concerned:

Node #	ASN of Node	Out-degree	known spammers
1	Everyone's Internet (AS 13749)	36,875	12
2	IQuest (AS 7332)	32,159	7
3	UUNet (AS 701)	31,682	5
4	UPC Broadband (AS 6830)	26,502	8
5	E-xpedient (AS 17054)	19,530	4

Table 1: AS numbers of hosts which have the highest out-degrees. The last column shows the number of hosts queried by this node that are known spammers (verified using logs from our spam sinkhole).

the pruned traffic amounts to less than 1% of the total DNSBL traffic. In this section, we present two surprising results: First, botnets are being used to perform DNSBL reconnaissance on behalf of bots in other botnets, which has implications for botnet detection. Second, the distribution of these queries across bots suggests that some DNSBL reconnaissance activities may be detectable in real-time, which has implications for early detection and mitigation.

Attempts to validate our hypotheses from Section 2 resulted in some interesting discoveries, including the discovery of new bots. We initially expected that most DNSBL lookups would be third-party lookups, as described in Section 2.2.1, and that we would be able to validate the queried nodes as being known bots. Instead, we discovered the opposite: the nodes with the highest values of λ_n in the pruned graph were *known* bots, while *the queried nodes in the graph were new, previously unknown bots*. Further, using data from our spam sinkhole [17], we found that some of these nodes were Windows machines and confirmed spam originators. This finding suggests that, in general, it may be possible to start with a set of known bots and use the DNSBL graph to "bootstrap" the discovery of new bots.

Table 1 shows five of the top queriers (*i.e.*, high out-degree nodes), *all* of which are known bots from our Bobax trace. Even more interesting is the fact that a few IP addresses queried by these nodes actually sent spam to our spam honeypot. Moreover, nearly all of IP addresses that sent spam to our honeypot were *not* present in our list of known bots. Due to the fact that our honeypot only captures a small portion of the Internet's spam, the fraction of total reconnaissance queries that we can confirm as spamming bots is small. Still, we believe it strongly suggests evidence of a known bot performing DNSBL reconnaissance on a distinct (and possibly newly compromised) botnet.

Figure 3 shows the distribution of out-degrees for all querying nodes present in the pruned DNSBL query graph. The long tail also confirms that bots already have the capability to distribute these queries, which is cause for concern. Our view of DNSBL queries is narrow (most querying nodes are geographically close to the DNSBL mirror), so we expect that more vantage points of DNSBL lookups would reveal other prominent "play-

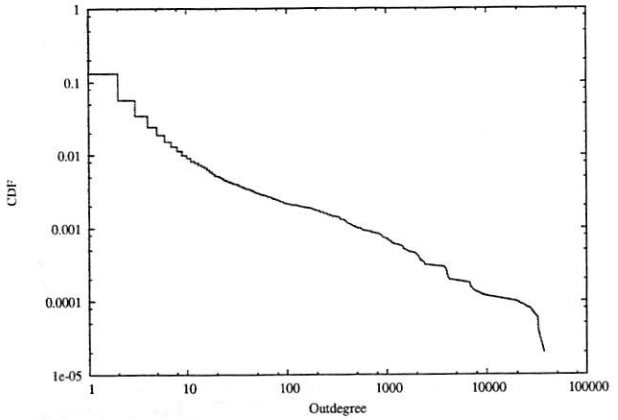

Figure 3: CDF of the distribution of out-degrees for querying IP addresses.

ers". The fact that the prominent players in our analysis were also bots suggests that these nodes may also be obvious candidates for the mitigation techniques described in Section 5.

5. Countermeasures

In Section 4, we found that the known bots in our Bobax trace were not the targets of lookups, but instead were issuing lookups for other, possibly newly compromised bots. This finding suggests a possible technique that could be used for the discovery of new bots, even without an initial list of suspects: an initial set of suspect IP addresses could be constructed by establishing a spam trap, which according to both previous work [17] and the observations in this paper, appear to be largely bots. Alternatively, a suspect node could be detected simply by identifying nodes in the DNSBL query graph with a high value of λ_n. Beginning with this initial suspect list, an operator may be able to conclude that, not only are the nodes that this node is querying likely bots, but also the node itself is likely a bot. If there are other high-degree nodes also querying the same bots, a detection algorithm might be able to "walk" the DNSBL graph (*e.g.*, from parent to parent) to discover multiple distinct botnets.

We believe that using such techniques to aggressively monitor botnet-based DNSBL reconnaissance may prove to be useful for mitigating spam: as noted in our previous work [17], most bots send a very low volume of spam to any single domain; thus, reporting a bot to blacklists *after* the spam is received may not be effective.

With the ability to distinguish reconnaissance queries from legitimate queries, a DNSBL operator might be able to mitigate spam more effectively. We speculate one possibility as follows: an operator could tune the behavior of the blackhole list server to mislead a botmaster, using a class of techniques we call *reconnaissance poisoning*. On one hand, the DNSBL could trick the botmaster into thinking that a particular bot was "clean" (*i.e.*, unlisted) when in fact it was listed, which would induce the botmaster to unwittingly send spam from blacklisted machines. On the other hand, the DNSBL could also reply to a reconnaissance query with an indication that a host was listed, even though it was not listed, thereby discouraging a botmaster from using a machine that would likely be capable of successfully sending spam.

Of course, active countermeasures such as reconnaissance poisoning do run the risk of false positives: if we mistakenly attribute a legitimate DNSBL query to a reconnaissance-based query, we could mislead a legitimate mail server into either mistakenly accepting spam that would have otherwise been rejected or, more regrettably, rejecting legitimate email. Such techniques could also be defeated if the botmaster queries multiple blacklist providers that maintain independent lists. Investigating the extent to which our detection metrics are subject to false positives, as well as the extent to which these false positives interfere with a legitimate mail server's filtering techniques, is part of our ongoing work.

6. Related Work

Botnets have been in use as vehicles of cybercrime for quite some time, but studies on how they spread, and techniques to counter them, are relatively scarce. Previous research has traced the history of botnets [18, 21, 22] and common modes of botnet operation [5]. This section briefly discusses previous botnet detection techniques and previous research on DNSBL traffic analysis.

Previous work has identified bots by examining the communication protocols used by botnets (*e.g.*, for "rallying"), most notably Internet Relay Chat (IRC) [7, 23]. Some have suggested the use of such protocols to identify and remediate botnets. For example, researchers have joined IRC-based botnets and enumerated victims using IRC commands [8]; others have used network traffic to identify IRC zombies [16]. Some researchers have identified bot victims by observing the unwanted traffic they generate, *e.g.*, the RST storms or backscatter generated by DDoS attacks using forged source addresses [15].

Studies show that many botnets are IRC-based [5, 22], though other protocols are being used [14]. Attempts have been made to detect such botnets using misuse-detection or basic intrusion detection analysis [3, 10]. Dagon *et al.* used DNS redirection to monitor botnets [6]. In contrast, the detection techniques described in this paper are more discreet because they do not require direct communication with any component of the botnet.

Jung *et al.* found that 80% of spam sources in their analysis were listed in at least one of seven popular blacklists [11], which correlates well with our independent previous study [17]. To the best of our knowledge, this paper presents the first study that uses direct analysis of DNSBL logs to infer other types of network behavior.

7. Conclusion

This paper has developed techniques and heuristics for detecting DNSBL reconnaissance activity, whereby botmasters perform lookups against the DNSBL to deter-

mine whether their spamming bots have been blacklisted. We first developed heuristics for counter-intelligence based on several possible ways we figured reconnaissance was being performed. We then studied the prevalence of each of these reconnaissance techniques. Much to our surprise, we found that bots were in fact performing reconnaissance on IP addresses for bots in other botnets. Based on this finding, we have outlined possibilities for new botnet detection techniques using a traversal of the DNSBL query graph, and we have suggested techniques that DNSBL operators might use to more effectively stem the spam originating from botnets. We are investigating the effectiveness of these detection and mitigation techniques as part of our ongoing work.

Acknowledgments

We thank Randy Bush, Wenke Lee, and Merrick Furst for feedback on some of the ideas in this paper. This work is supported in part by NSF grant CCR-0133629 and Office of Naval Research grant N000140410735. The contents of this work are solely the responsibility of the authors and do not necessarily represent the official views of NSF or the U.S. Navy. Data used in this paper was obtained as part of an information disclosure granted by the Georgia Tech Research Corporation, ref. Record of Invention GTRC ID 3828. Please consult the authors regarding its citation or use.

Notes

[1] DNSBL queries issued by mail servers are often performed by directly querying the DNSBL, rather than relying on a local resolver. For example, SpamAssassin [20] implements its own recursive DNS resolver. Hosts performing reconnaissance are also unlikely to query DNSBLs using local resolvers. Thus, in both cases, the querying IP address observed at the DNSBL correctly reflects the end-host performing the query.

[2] This heuristic assumes that networks generally use the same host for both inbound and outbound mail servers. Although this configuration is common, some large networks separate the hosts responsible for inbound and outbound mail servers. In this case, queries from the inbound mail server might be misinterpreted as a reconnaissance attempt.

[3] When $d_{n,in}$ is zero (which is commonly the case), we can simply consider λ_n to be a very large number.

REFERENCES

[1] Bobax trojan analysis. `http://www.lurhq.com/bobax.html`, March 2005.

[2] Symantec Security Alert–W32.Bobax.D worm. `http://www.sarc.com/avcenter/venc/data/w32.bobax.d.html`.

[3] D. Brumley. Tracking hackers on IRC. `http://www.doomdead.com/texts/ircmirc/TrackingHackersonIRC.htm`, 2003.

[4] CNN Technology News. Expert: Botnets No. 1 emerging Internet threat. `http://www.cnn.com/2006/TECH/internet/01/31/furst/`, Jan. 2006.

[5] E. Cooke, F. Jahanian, and D. McPherson. The Zombie Roundup: Understanding, Detecting and Disrupting Botnets. In *Usenix Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, June 2005.

[6] D. Dagon, C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS '06)*, 2006.

[7] S. Dietrich, N. Long, and D. Dittrich. Analyzing distributed denial of service attack tools: The shaft case. In *Proceedings of the LISA 2000 System Administration Conference*, December 2000.

[8] F. C. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. Technical Report ISSN-0935-3232, RWTH Aachen, April 2005.

[9] L. H. Gomes, C. Cazita, J. Almeida, V. Almeida, and W. Meira. Characterizing a Spam Traffic. In *Proc. ACM SIGCOMM Internet Measurement Conference*, Taormina, Sicily, Italy, Oct. 2004.

[10] C. Hanna. Using snort to detect rogue IRC bot programs. Technical report, October 2004.

[11] J. Jung and E. Sit. An Empirical Study of Spam Traffic and the Use of DNS Black Lists. In *Proc. ACM SIGCOMM Internet Measurement Conference*, pages 370–375, Taormina, Sicily, Italy, Oct. 2004.

[12] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[13] S. Krasser, G. Conti, J. Grizzard, J. Gribschaw, and H. Owen. Real-time and forensic network data analysis using animated and coordinated visualization. In *Proceedings of the 6th IEEE Information Assurance Workshop*, 2005.

[14] B. Krebs. Bringing botnets outof the shadows. `http://www.washingtonpost.com/wp-dyn/content/article/2006/03/21/AR2006032100279.html`, 2006.

[15] D. Moore, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. In *Proceedings of the 2001 USENIX Security Symposium*, 2001.

[16] S. Racine. Analysis of internet relay chat usage by ddos zombies. `ftp://www.tik.ee.ethz.ch/pub/students/2003-2004-Wi/MA-2004-01.pdf`, 2004.

[17] A. Ramachandran and N. Feamster. Understanding the Network-Level Behavior of Spammers. In *Proc. ACM SIGCOMM*, Pisa, Italy, Sept. 2006.

[18] P. Ramneek. Bots & Botnets: An Overview. `http://www.giac.com/practical/GSEC/Ramneek_Puri_GSEC.pdf`, 2003.

[19] S. Schechter and M. Smith. Access for sale. In *2003 ACM Workshop on Rapid Malcode (WORM'03)*. ACM SIGSAC, October 2003.

[20] SpamAssassin, 2005. `http://www.spamassassin.org/`.

[21] SwatIt. Bots, drones, zombies, worms and other things that go bump in the night. `http://swatit.org/bots/`, 2004.

[22] Virus Bulletin 2005 Paper on 'Bots and Botnets'. `http://arachnid.homeip.net/papers/VB2005-Bots_and_Botnets-1.0.2.pdf`.

[23] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

Leveraging Good Intentions to Reduce Unwanted Network Traffic

Marianne Shaw
marianne.shaw@gmail.com

Abstract

We present a solution to reduce unwanted network traffic by enabling either side of a conversation to summarily terminate the conversation without the other endpoint's cooperation. Our work is motivated by the observation that many compromised endhosts on the network are well-intentioned but easily compromised; these machines are often compromised and their resources used to attack others. We argue that the good intentions of these endhosts can be leveraged to construct a control plane that ensures that, even when compromised, these well-intentioned machines only generate well-behaved traffic. This independently enforced control plane prevents an endhost from blatantly disregarding requests to cease traffic generation. The solution's viability rests upon its unobtrusive deployment. No extra mechanism is needed within the network as all enforcement is performed at the endhosts. Hosts are not restricted in their behavior except by the behavior demanded by their peers.

1 Introduction

Commodity PCs and broadband have enabled huge numbers of users to connect to the Internet. Once connected to the Internet, user-administered machines are bombarded with attacks aimed at gaining control of their physical resources. Compromised machines are used to propagate worms and viruses, participate in DDoS attacks, provide services such as spam relays or IRC servers, and be part of organized botnets. Users do not want their machines to be compromised and used for malicious purposes, but they do not have the knowledge or skill to prevent it. This work asks, can we leverage the users' non-malicious intentions to prevent their machines from being used to generate unwanted traffic?

This work is driven by several key observations. First, we accept that a machine will be compromised and attempts will be made to use it to generate malicious network traffic. Attackers use compromised machines to amplify their ability to inflict damage; we can inhibit their potential impact by reducing the benefits of incorporating these machines. Next, we believe that many user-administrators do not want their machines to be used to inflict damage, and they would be willing to thwart such activity if they could. There is benefit to preventing the injection of malicious traffic into the Internet, rather than trying to deal with it once it is inside the network. We can leverage users' good intentions to co-locate an enforcement mechanism with a host and use it to prevent the injection of certain traffic into the network. Finally, defining and identifying unwanted behavior is difficult and often subjective; two hosts may not classify the same traffic in the same way. Rather than propose a universal definition of good or bad traffic, we seek to provide a way for traffic recipients to request the temporary cessation of traffic that they themselves deem undesirable.

Our solution puts control of the network packet exchange between two hosts in the hands of both of the endpoints in such a way that each endpoint has complete control over it. We do this by constructing an independent control plane that is co-located with each well-meaning host. When one endpoint requests that the other temporarily stops sending traffic to it, the control plane prevents that second host from disregarding the termination request; all outgoing packets are immediately dropped at the generating host and never enter the Internet. This conversation "ripcord" is necessary because a compromised host may have an altered networking stack or operating system [11] which ignores all (if they exist) standard termination requests. As long as both hosts believe that the conversation is well-behaved, there is minimal impact on the conversation.

2 Approach

Computers connected to the Internet continue to be attacked and compromised. User-administered machines are especially vulnerable to compromise as they often run unpatched computer programs that allow attackers to capitalize upon well-documented software flaws. Symantec [23] found that several popular unpatched desktop operating systems were compromised within 1.5 hours of being connected to the Internet. Once compromised, the physical resources of a machine may be used to further propagate an attack or may be incorporated into a network of attack machines.

We seek to prevent well-intentioned machines from

being compromised and used to amplify an attacker's ability to inflict damage on the Internet. To do this, we make it possible for a host to tell a machine from which it is currently receiving packets to temporarily stop sending packets, and to reasonably expect that the request will be honored. Thus, if a compromised host is sending network attack traffic, a receiving host can stop the incoming packets to protect itself. While this work focuses on enabling endhosts to request the cessation of an attack, this mechanism could be used by overloaded hosts to temporarily delay incoming traffic.

There are a few key requirements for achieving our goals. 1) Upon receiving unwanted network traffic, a host must be able to identify the source of the traffic to which it can send a termination request. Any host that voluntarily adopts our mechanism must therefore be prevented from spoofing its packets' source address. 2) We only honor requests to temporarily terminate an existing packet stream. We do not allow hosts to proactively blacklist traffic that they have not yet received; nor do we honor termination requests for packet streams that have been inactive for a long duration. 3) Only a recipient of unwanted network traffic can request that a packet stream be terminated; malicious hosts should not be able to use this mechanism to force a well-intentioned host into silence. 4) Our enforcement mechanism must be voluntarily adopted by endhosts. We cannot rely upon the introduction of new mechanism within the network itself. We cannot impose undue restrictions upon the network services used by the host simply to accommodate our mechanism. 5) Upon receiving a termination request, we must be able to terminate a packet stream without the receiving machine's cooperation. Once a well-intentioned machine has been compromised, rootkits [9] can be used to gain superuser privileges on a machine; the machine's networking stack and OS can then be modified, replaced, or subverted [11]. Typically, a machine's OS and networking stack enforce well-established "good behavior;" once these modules are compromised, a machine is able to blatantly disregard all standard networking conventions.

We leverage the good intentions of non-malicious users to co-locate an enforcement mechanism with each host. The mechanism itself must be independent, not network-addressable, and able to interpose on all traffic in to and out of the host. Provided it can meet all of these specifications, the mechanism itself can be implemented in either hardware, software, or a combination of both.

Although each mechanism only enforces our requirements for an individual host, in aggregate these mechanisms create an independent control plane. This plane ensures that all traffic that it allows to enter the Internet can be summarily terminated at a recipient's request. Recipients of unwanted network traffic now have a course of action by which they can protect themselves without requiring the cooperation of their ISP.

3 Design

Our control-plane enforcement mechanisms must ensure that potential victims can accurately identify their attackers from the offending packet stream, determine the validity of requests to temporarily stop an existing packet stream, and enforce valid network traffic termination requests without endhost cooperation. A combination of control plane signalling and our enforcement mechanism make this possible.

3.1 Control plane signalling

To leverage users' good intentions, our approach must be able to provide significant benefits without introducing a system administration burden on the users. Once a user allows us to interpose on all network traffic in to and out of their machine, all necessary information should be gleaned from the traffic that we observe. From the stream of packets, we must be able to identify the co-located host's unique identifier, the start and end of a conversation between two hosts, and a termination request.

Unique Identifier Ideally, the Internet would provide each host with a unique non-forgeable identifier that could be used to provide accountability for actions taken by a networked host. In practice, hosts have the ability to transmit arbitrary network packets; they can assume another host's identity by transmitting packets with spoofed source addresses. Additionally, rather than having a single assigned static IP address, many hosts dynamically acquire their IP address for a short period of time.

Accountability is a necessary element of our solution. A recipient of unwanted traffic must be able to identify and contact the host that sent the traffic; at the same time, that host should not be penalized for spoofed packets sent by other machines. We must ensure that well-intentioned hosts cannot send packets with spoofed source addresses, but we must also monitor the packets that our host did send to deny accountability for others' actions.

A host's unique identifier is therefore the source IP address that it used to send a stream of packets during a specific time period. This approach, while not perfect, is sufficient for our purposes because we only honor valid termination requests for active traffic streams.

Our enforcement mechanism can determine a host's unique identifier from the consistent source address of the packets that it sends. We can leverage events that signify an expected change in IP address to recognize valid IP address changes. For example, hosts that dynamically acquire IP addresses tend to exhibit lulls in their network activity before they acquire a new IP address. Alternatively, if the control mechanism is directly connected to the host's network card, it can detect when

a card is reset by the link going down. These events occur over a period of seconds. In contrast, many network attacks rapidly send packets with quickly changing spoofed source addresses; our mechanism should characterize these as spoofed packets and drop them.

We can prevent our host from being penalized for spoofed packets sent by other machines by tracking the packets that were actually sent by the host. Rather than log each individual packet transmitted, we can track the fact that we sent packets associated with a particular network conversation (defined below) during a certain time frame. When presented with a termination request for a packet that the host did not send, the enforcement mechanism simply discards the request. Because termination requests are only honored for active packet streams, the amount of state required is bounded by the number of currently active streams.

Defining a network conversation A network conversation defines both the criteria and the granularity that we use to track sequences of network packets; it dictates which packets will be dropped when a termination request is received. Hosts receiving unwanted traffic must weigh the cost of receiving those incoming packets against the cost of terminating the network conversation.

We can provide the ability for a host to summarily terminate a conversation for many different definitions of a network conversation provided we are able to uniquely identify the principals of a conversation from each packet sent by the host, identify the start and stop of a conversation by observing the packet stream, and identify the termination request associated with a conversation.

Conversation principals Traditionally, IP source and destination addresses have been the basis for identifying network conversations. Additional properties such as IP protocol and source and destination ports have been used to refine these principals. We can extend this set to include more coarse grained principals. IP prefixes could be used instead of IP addresses; thus, our enforcement mechanism can honor requests to drop all UDP port 666 traffic destined for 10.10.10.*. Alternatively, a host can request that it no longer be sent any TCP traffic.

Conversation start/stop The enforcement mechanism must know exactly what indicates that a conversation is active and inactive. Ideally, we can identify the start and stop of a conversation simply by observing the contents of network packets and maintaining internal state. For example, TCP uses explicit start, stop, and termination sequences for maintaining connections. We can use this protocol signalling to restrict and terminate wayward conversations; a prototype for TCP is outlined in Section 4.

However, for many conversations there is no explicit signalling indicating the conversation delimiters, and we must infer the start and stop of the conversation by observing patterns of network activity. Correctly inferring these endpoints can be difficult; although we can use the existence of network traffic between two principals to recognize that a conversation is active, for many long-lived conversations we cannot use the absence of network traffic to determine that a conversation has been stopped.

Termination requests Hosts require an explicit signalling mechanism for terminating a network conversation. In addition to indicating which network conversation is being terminated, these requests must either indicate or imply the amount of time during which packets must not be sent; we do not allow network conversations to be terminated indefinitely.

Certain protocols may have existing support for terminating a conversation; for example, TCP uses RST packets to reset a conversation. However, if we must infer active conversations based upon the existence of network traffic between two principals, it is unlikely that there will already be an existing explicit signal for terminating the conversation. To accommodate these ad-hoc definitions of conversations and enable hosts to reliably terminate them, it may be necessary to provide a new signalling mechanism.

Termination requests must demonstrate that the request is being sent by the recipient of the unwanted network traffic; spoofed termination requests should be discarded. The requesting host can be authenticated through the exchange of a large random nonce with the enforcement mechanism. If the nonce cannot be overlaid on top of a network conversation's existing protocol, then an explicit authenticating nonce exchange may be required. Once successfully exchanged, the nonce can be injected into a termination request to establish its authenticity.

3.2 Enforcement mechanism

To convince well-intentioned users to allow our enforcement mechanisms to interpose on their machine's network traffic, we must be unobtrusive yet effective at preventing their machines from attacking other hosts.

The enforcement mechanism cannot be bypassed or subverted by attackers The enforcement mechanism must interpose on all traffic in to and out of a machine, and it must remain completely isolated and independent from that machine. If the enforcement mechanism is not independent, when the host machine is compromised the attacker can simply "turn off" all packet-restricting components. Incapacitating the enforcement mechanism should require physical access to the host machine to prevent it being silently disabled by anonymous attackers.

The enforcement mechanism must actively participate in each conversation that it may need to forcibly terminate. This work aims to reduce attack traffic that is generated by a compromised host; in this scenario, *both* sides of the enforcement mechanism are controlled by the at-

tacker. If the enforcement mechanism does not inject itself into a packet stream, the compromised machine can collude with an external attacker to prevent a conversation's termination.

Actively injecting a nonce into a packet stream enables the enforcement mechanism to independently authenticate an endpoint. Only hosts directly on the path taken by outgoing network packets will be able to reliably establish, maintain, or terminate a conversation.

The enforcement mechanism cannot be undermined by replaying a previous conversation through the mechanism This is especially important as many hosts acquire their IP addresses dynamically; an attacker could try to replay a previous conversation to inflict damage on the host now allocated a specific IP address. Therefore, the enforcement mechanism must require proof of "liveness" for all conversations flowing through it. The nonces used to authenticate endpoints should be randomly generated at the time they are needed.

The enforcement mechanism can be deployed incrementally by end users and removed as needed, which should be extremely rare. The enforcement mechanism must be effective at reducing unwanted network traffic as it is incrementally deployed. Not all user-administered machines are going to immediately install a mechanism that prevents their machines from being used to attack others; indeed, not all users will want to install such a mechanism. The enforcement mechanism must not rely upon upgraded hardware within the network or widespread deployment and adoption of new protocols. By co-locating the enforcement mechanism with the hosts that they are potentially restricting, our solution can be deployed by individual users without requiring ambitious network hardware or software upgrades.

4 TCP Prototype

We describe a prototype implementation of our solution for TCP. Because TCP is a connection-oriented protocol, we were able to use its existing characteristics to develop a prototype that is virtually invisible to end-hosts. Our enforcement mechanism executes on a separate physical machine whose sole purpose is to act as a gateway between our user-administered host and the larger network. All traffic to and from the host must pass through our enforcement mechanism over the dedicated Ethernet connection.

The system "learns" the host's IP address by observing the source IP address in all outgoing network packets. If the network link goes down or if there is a sustained period of network inactivity, the system re-learns the IP address when outgoing packets are observed. All outgoing packets using a different source IP address are dropped.

We define our network conversation to be the connection established between two (IP:port) pairs using TCP's three-way handshake protocol. We leverage TCP's handshake protocol to determine the start of a network conversation. TCP also contains two distinct techniques for closing a connection: a FIN-ACK sequence initiated by each half of the connection, and a RST packet sent by either side of the connection. As with TCP's connection establishment, we simply leverage this explicit signalling to track the end of a conversation.

As long as neither host is compromised or misbehaving, TCP's built-in control signalling ensures that hosts can terminate any undesired connection. The true merit of our enforcement mechanism is observed when one of the hosts is ignoring the TCP termination messages that it receives.

Imagine that a remote host establishes a TCP connection with our local host, and the local host starts flooding the remote host with network packets. The remote host may send a RST packet to stop the packet flood, but the local host may simply ignore the RST packet and continue to send high rates of unwanted packets. Our enforcement mechanism monitors each established connection to prevent this type of scenario. Once it observes a valid incoming RST packet, the enforcement mechanism drops all outgoing network packets associated with this connection.

In its efforts to restrict unwanted outgoing traffic, the enforcement mechanism must be careful not to allow spoofed RST packets to cause it to incorrectly terminate TCP connections. Additionally, it must prevent a compromised host and a colluding remote attacker from spoofing a connection establishment sequence and using it to attack a third network host. TCP uses the exchange of sequence numbers to provide reasonably good authentication of each endpoint during connection establishment and teardown. This approach, however, relies upon the belief that at least one of the participating hosts is well-behaved and trustworthy. Because both sides of our enforcement mechanism are potentially compromised, it cannot rely upon the validity of TCP's authentication.

Our enforcement mechanism must provide its own endpoint authentication; it does this by adding a random 32-bit nonce to the initial sequence number (ISN) provided by each host during connection establishment. By adding this random value to each host's sequence number, the enforcement mechanism authenticates each endpoint when the modified sequence number is returned. This is the same authentication technique used by standard TCP, but when used by the enforcement mechanism it ensures that two untrusted, colluding hosts cannot subvert the enforcement mechanism using pre-established ISNs. The random nonces are individually generated for each connection establishment seen by the enforcement

mechanism; thus the nonce provides the "liveness" property necessary for thwarting replay attacks.

Adding the authenticating nonce to TCP's sequence number requires the enforcement layer to continue modifying all subsequent packets' sequence numbers. It must add the nonce to all outgoing packets' sequence numbers and remove the nonce from all incoming packets' sequence numbers. The enforcement mechanism must also recalculate the checksum for each modified packet; the checksum does not need to be completely recalculated but can simply be updated by the difference between the old and new fields.

The enforcement mechanism maintains per-connection state to track the status of each TCP connection. Our implementation required 108 bytes of connection state for each active connection. Because our mechanism enforces conversation termination for a single host, the number of active connections that we are monitoring should remain small, as will our overall storage requirements.

5 Related Work

A diverse set of techniques and mechanisms have been proposed to address the widespread, damaging nature of modern Internet attacks.

A variety of projects have attempted to characterize network traffic; these characterizations can then be used to filter or identify unwanted network traffic. Network connectivity patterns have been used to characterize both normal and abnormal network traffic ([3], [25], [22], [24]) including the propagation patterns of individual worms ([15], [14], [28].) Studies have quantified denial-of-service [16] activity and spyware [19] seen on the Internet. Worms signatures ([10], [17], [21]) can be used in the identification of traffic containing worms. Other work has focused on the characteristics of "normal" traffic ([12], [27]) for detecting or rate-limiting anomalous behavior.

Our proposed solution is orthogonal to this work in that we require an endhost to determine for itself whether or not a particular stream of network traffic is unwanted. We provide a mechanism whereby the host can request that a packet stream be halted; a host can leverage any of these characterization techniques to decide if the stream is unwanted.

Existing research has proposed introducing new mechanism in the network to identify, account for, and eliminate unwanted traffic. IP Traceback [20] uses network state to identify the path taken by unwanted DoS traffic. Pushback [8] and AITF [2] install packet filters at routers within the network to filter out unwanted traffic. Capability-based networks [1] use packet processing hardware at trust boundaries to enable hosts to communicate while network attacks occur. In the face of congestion, network hardware may selectively mark [6] or drop packets associated with high packet rates.

In contrast with these network-based mechanisms, this work proposes a mechanism that is co-located with a host to prevent unwanted network traffic from being injected into the Internet. We deploy our enforcement mechanisms at potentially malicious traffic sources so that we can drop unwanted traffic before it can impact other hosts. This principle is similarly embraced by network ingress filtering [5], reverse firewalls [13], and IP throttling [26]. Although we allow hosts to push cessation requests upstream like Pushback [8] and AITF [2], we depend upon users' willingness to have mechanism co-located with their hosts to eliminate their need for increased mechanism in the network. By leveraging hardware mechanism at the source of network traffic, our solution can be incrementally deployed at the endhosts. No large-scale network hardware or software upgrades are required before benefits can begin to accrue.

Unlike many source-limiting approaches, our solution does not merely enforce a well-established definition of good behavior, such as limiting the rates of outgoing connections, packets, or source IP addresses. Our work leverages techniques that use feedback mechanisms to indicate when a host is behaving poorly. RED [6] and ECN Nonce [4] use network mechanisms to inform a host that there is network congestion; and TCP uses packet drops to scale back its transmission rate. Our work differs from these approaches in that we only rely upon endhosts to provide negative feedback in the form of requests to terminate malicious conversations.

Finally, a key property of our solution is the independence of our enforcement mechanism. Our enforcement mechanism assumes that it is surrounded by untrustworthy, malicious entities that will try to subvert or disable it. Therefore, the enforcement mechanism must be active in its efforts to prevent malicious traffic. This is in direct contrast to many firewalls [7] and intrusion detection systems [18] which assume that at least one side of the enforcement mechanism is trustworthy.

5.1 Summary

We have argued that we can leverage the good intentions of users to reduce unwanted traffic on the Internet. User-administrated machines are frequently vulnerable to compromise, and once compromised their physical resources can be used to attack other hosts. By co-locating an enforcement mechanism with these well-intentioned hosts, recipients of unwanted traffic can summarily terminate streams of incoming packets from these hosts. If a host has been compromised and is attacking other machines, the victims have the ability to, at least temporarily, terminate the attack.

References

[1] Tom Anderson, Timothy Roscoe, and David Wetherall. Preventing internet denial-of-service with capabilities. *SIGCOMM Comput. Commun. Rev.*, 34(1):39–44, 2004.

[2] Katerina J. Argyraki and David R. Cheriton. Active internet traffic filtering: Real-time response to denial of service attacks. *CoRR*, cs.NI/0309054, 2003.

[3] Daniel R. Ellis, John G. Aiken, Kira S. Attwood, and Scott D. Tenaglia. A behavioral approach to worm detection. In *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 43–53, New York, NY, USA, 2004. ACM Press.

[4] David Ely, Neil Spring, David Wetherall, Stefan Savage, and Tom Anderson. Robust congestion signaling. In *ICNP*, 2001.

[5] Paul Ferguson and Daniel Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing RFC 2267. IETF RFC Publication, January 1998.

[6] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[7] Michael B. Greenwald, Sandeep K. Singhal, Jonathan R. Stone, and David R. Cheriton. Designing an academic firewall: Policy, practice, and experience with surf. In *SNDSS '96: Proceedings of the 1996 Symposium on Network and Distributed System Security (SNDSS '96)*, page 79, Washington, DC, USA, 1996. IEEE Computer Society.

[8] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *NDSS*, 2002.

[9] Nick L. Petroni Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In *USENIX Security Symposium*, pages 179–194, 2004.

[10] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *USENIX Security Symposium*, pages 271–286, 2004.

[11] Samuel T. King, Peter M. Chen, Yi-Min Wang, Chad Verbowski, Helen J. Wang, and Jacob R. Lorch. Subvirt: Implementing malware with virtual machines. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*, page 65, Washington, DC, USA, 1997. IEEE Computer Society.

[12] John McHugh and Carrie Gates. Locality: a new paradigm for thinking about normal behavior and outsider threat. In *NSPW '03: Proceedings of the 2003 workshop on New security paradigms*, pages 3–10, New York, NY, USA, 2003. ACM Press.

[13] Jelena Mirkovic, Gregory Prier, and Peter L. Reiher. Attacking ddos at the source. In *ICNP '02: Proceedings of the 10th IEEE International Conference on Network Protocols*, pages 312–321, Washington, DC, USA, 2002. IEEE Computer Society.

[14] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The spread of the sapphire/slammer worm. Technical report, CAIDA, ICSI, Silicon Defense, UC Berkeley EECS and UC San Diego CSE, 2003.

[15] D. Moore, C. Shannon, and J. Brown. Code-Red: a case study on the spread and victims of an Internet worm. In *Proc. of Internet Measurement Workshop 2002*, Nov 2002.

[16] David Moore, Geoffrey M. Voelker, and Stefan Savage. Inferring Internet denial-of-service activity. In *USENIX Security Symposium*, 2001.

[17] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy*, 2005.

[18] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.

[19] Stefan Saroiu, Steven D. Gribble, and Henry M. Levy. Measurement and analysis of spyware in a university environment. In *NSDI*, pages 141–153, 2004.

[20] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for ip traceback. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 295–306, New York, NY, USA, 2000. ACM Press.

[21] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In *OSDI*, pages 45–60, 2004.

[22] S. Staniford-Chen et al. GrIDS—A graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, volume 1, pages 361–370, October 1996.

[23] Symantec. Symantec internet security threat report, volume ix, Mar 2006.

[24] Godfrey Tan, Massimiliano Poletto, John Guttag, and Frans Kaashoek. Role Classification of Hosts within Enterprise Networks Based on Connection Patterns. In *The USENIX Annual Technical Conference 2003*, San Antonio, TX, June 2003.

[25] T. Toth and C. Kruegel. Connection-history based anomaly detection, 2002.

[26] J. Twycross and M.M. Williamson. Implementing and testing a virus throttle. In *USENIX Security Symposium*, 2003.

[27] Matthew M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference*, page 61, Washington, DC, USA, 2002. IEEE Computer Society.

[28] Cynthia Wong, Stan Bielski, Jonathan M. McCune, and Chenxi Wang. A study of mass-mailing worms. In *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 1–10, New York, NY, USA, 2004. ACM Press.

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of system administrators, developers, programmers, and engineers working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering technical excellence and innovation
- encouraging computing outreach in the community at large
- providing a neutral forum for the discussion of critical issues

Membership Benefits

- Free subscription to *;login:*, the Association's magazine, both in print and online
- Online access to all Conference Proceedings from 1993 to the present
- Access to the USENIX Jobs Board: Perfect for those who are looking for work or are looking to hire from the talented pool of USENIX members
- The right to vote in USENIX Association elections
- Discounts on technical sessions registration fees for all USENIX-sponsored and co-sponsored events
- Discounts on purchasing printed Proceedings, CD-ROMs, and other Association publications
- Discounts on industry-related publications: see http://www.usenix.org/membership/specialdisc.html

For more information about membership, conferences, or publications, see http://www.usenix.org.

SAGE, a USENIX Special Interest Group

SAGE is a Special Interest Group of the USENIX Association. Its goal is to serve the system administration community by:

- Establishing standards of professional excellence and recognizing those who attain them
- Promoting activities that advance the state of the art or the community
- Providing tools, information, and services to assist system administrators and their organizations
- Offering conferences and training to enhance the technical and managerial capabilities of members of the profession

Find out more about SAGE at http://www.sage.org.

Thanks to USENIX & SAGE Supporting Members

Addison-Wesley Professional/ Prentice Hall Professional
Ajava Systems, Inc.
AMD
Asian Development Bank
Cambridge Computer Services, Inc.
EAGLE Software, Inc.
Electronic Frontier Foundation
Eli Research
FOTO SEARCH Stock Footage and Stock Photography
GroundWork Open Source Solutions
Hewlett-Packard
IBM
Infosys
Intel
Interhack
The Measurement Factory
Microsoft Research
MSB Associates
NetApp
Oracle
OSDL
Raytheon
Ripe NCC
Sendmail, Inc.
Splunk
Sun Microsystems, Inc.
Taos
Tellme Networks
UUNET Technologies, Inc.

ISBN 1-931971-46-3